Shinken Documentation

Release 1.4

Shinken Team

August 07, 2015

Contents

1 About **3**
1.1 About Shinken . . . 4
1.2 Feature comparison between Shinken and Nagios . . . 5
1.3 Shinken notable innovations . . . 5
1.4 The project Vision . . . 7
1.5 Feature selection and release cycle . . . 8
1.6 Release code names . . . 8

2 Getting Started **9**
2.1 Advice for Beginners . . . 10
2.2 Quickstart Installation Guides . . . 10
2.3 Installations . . . 12
2.4 Upgrading Shinken . . . 33
2.5 Monitoring Windows Machines . . . 34
2.6 Monitoring GNU/Linux & Unix Machines . . . 39
2.7 Monitoring Network Printers . . . 40
2.8 Monitoring Routers and Switches . . . 43
2.9 Monitoring Publicly Available Services . . . 47
2.10 Nagios/Shinken Plugins . . . 53
2.11 Introduction . . . 53
2.12 What Are Plugins? . . . 53
2.13 Shinken integrated data acquisition modules . . . 53
2.14 Plugins As An Abstraction Layer . . . 54
2.15 What Plugins Are Available? . . . 54
2.16 Obtaining Plugins . . . 55
2.17 How Do I Use Plugin X? . . . 55
2.18 Plugin API . . . 55

3 Configuring Shinken **57**
3.1 Configuration Overview . . . 58
3.2 Main Configuration File Options . . . 59
3.3 Object Configuration Overview . . . 70
3.4 Object Definitions . . . 72
3.5 Custom Object Variables . . . 73
3.6 Main advanced configuration . . . 75

4 Running Shinken **89**
4.1 Verifying Your Configuration . . . 90

4.2 Starting and Stopping Shinken . . . 90

5 The Basics **93**
5.1 Setting up a basic Shinken Configuration . . . 94
5.2 Nagios/Shinken Plugins . . . 99
5.3 Understanding Macros and How They Work . . . 101
5.4 Standard Macros in Shinken . . . 105
5.5 Host Checks . . . 113
5.6 Service Checks . . . 115
5.7 Active Checks . . . 116
5.8 Passive Checks . . . 118
5.9 State Types . . . 121
5.10 Time Periods . . . 123
5.11 Determining Status and Reachability of Network Hosts . . . 125
5.12 Notifications . . . 131
5.13 Active data acquisition modules . . . 134
5.14 Network dependencies . . . 136
5.15 Logical dependencies . . . 141
5.16 Update Shinken . . . 142

6 Medium **145**
6.1 Business rules . . . 146
6.2 Monitoring a DMZ . . . 149
6.3 Shinken High Availability . . . 150
6.4 Mixed GNU/linux AND Windows pollers . . . 152
6.5 Notifications and escalations . . . 153
6.6 The Notification Ways, AKA mail 24x7, SMS only the night for a same contact . . . 157
6.7 Passive data acquisition . . . 158

7 Advanced Topics **161**
7.1 External Commands . . . 162
7.2 Event Handlers . . . 163
7.3 Volatile Services . . . 167
7.4 Service and Host Freshness Checks . . . 169
7.5 Distributed Monitoring . . . 170
7.6 Redundant and Failover Network Monitoring . . . 179
7.7 Detection and Handling of State Flapping . . . 179
7.8 Notification Escalations . . . 183
7.9 On-Call Rotations . . . 188
7.10 Monitoring Service and Host Clusters . . . 192
7.11 Host and Service Dependencies . . . 192
7.12 State Stalking . . . 198
7.13 Performance Data . . . 199
7.14 Scheduled Downtime . . . 201
7.15 Adaptive Monitoring . . . 202
7.16 Predictive Dependency Checks . . . 203
7.17 Cached Checks . . . 205
7.18 Passive Host State Translation . . . 209
7.19 Service and Host Check Scheduling . . . 209
7.20 Object Inheritance . . . 209
7.21 Advanced tricks . . . 218
7.22 Business rules . . . 225
7.23 Migrating from Nagios to Shinken . . . 228
7.24 Multi layer discovery . . . 228

7.25 Multiple action urls . . . 231
7.26 Aggregation rule . . . 232
7.27 Scaling Shinken for large deployments . . . 233
7.28 Defining advanced service dependencies . . . 237
7.29 Shinken's distributed architecture . . . 242
7.30 Shinken's distributed architecture with realms . . . 244
7.31 Macro modulations . . . 246
7.32 Shinken and Android . . . 247
7.33 Send sms by gateway . . . 250
7.34 Triggers . . . 251
7.35 Unused nagios parameters . . . 253
7.36 Advanced discovery with Shinken . . . 261
7.37 Discovery with Shinken . . . 264

8 Config **267**
8.1 Host Definition . . . 268
8.2 Host Group Definition . . . 275
8.3 Service Definition . . . 276
8.4 Service Group Definition . . . 283
8.5 Contact Definition . . . 285
8.6 Contact Group Definition . . . 288
8.7 Time Period Definition . . . 289
8.8 Command Definition . . . 291
8.9 Service Dependency Definition . . . 293
8.10 Service Escalation Definition . . . 295
8.11 Host Dependency Definition . . . 297
8.12 Host Escalation Definition . . . 299
8.13 Extended Host Information Definition . . . 301
8.14 Extended Service Information Definition . . . 303
8.15 Notification Way Definition . . . 304
8.16 Realm Definition . . . 306
8.17 Arbiter Definition . . . 307
8.18 Scheduler Definition . . . 308
8.19 Poller Definition . . . 310
8.20 Reactionner Definition . . . 311
8.21 Broker Definition . . . 313

9 Shinken Architecture **315**
9.1 Arbiter supervision of Shinken processes . . . 316
9.2 Advanced architectures . . . 318
9.3 How are commands and configurations managed in Shinken . . . 323
9.4 Problems and impacts correlation management . . . 325
9.5 Problems and impacts correlation management . . . 326
9.6 Shinken Architecture . . . 328

10 Troubleshooting **331**
10.1 FAQ - Shinken troubleshooting . . . 332

11 Integration With Other Software **337**
11.1 Integration Overview . . . 338
11.2 SNMP Trap Integration . . . 339
11.3 TCP Wrappers Integration . . . 340
11.4 Thruk . . . 341
11.5 Nagios CGI UI . . . 344
11.6 Thruk interface . . . 345

11.7 Use Shinken with ... 345
11.8 Use Shinken with Centreon 347
11.9 Use Shinken with Graphite 349
11.10 Use Shinken with Multisite 351
11.11 Use Shinken with Nagvis 352
11.12 Use Shinken with Old CGI and VShell 353
11.13 Use Shinken with PNP4Nagios 354
11.14 Use Shinken with Thruk 356
11.15 Use Shinken with WebUI 358

12 Security and Performance Tuning 363
12.1 Security Considerations 364
12.2 Tuning Shinken For Maximum Performance 365
12.3 Scaling a Shinken installation 366
12.4 Shinken performance statistics 366

13 How to monitor ... 367
13.1 Monitoring an Asterisk server 368
13.2 check_wmi_plus.pl for shinken on windows 370
13.3 Monitoring Active Directory 372
13.4 Monitoring a DHCP server 374
13.5 Monitoring Microsoft Exchange 375
13.6 Monitoring a IIS server 378
13.7 Monitoring Linux Devices 380
13.8 Monitoring Linux Devices 383
13.9 Monitoring a Linux via a Local Agent 384
13.10 Monitoring a Linux via SNMP 384
13.11 Monitoring Publicly Available Services 387
13.12 Monitoring a printer 391
13.13 Monitoring Network Devices 393
13.14 Monitoring Windows Devices 396
13.15 Monitoring Microsoft Mssql server 399
13.16 Monitoring MySQL 402
13.17 Monitoring VMware Machines 405
13.18 Monitoring Microsoft Mssql server 408
13.19 Monitoring MySQL 411
13.20 Monitoring Publicly Available Services 414
13.21 Monitoring an Oracle database server 417
13.22 Monitoring a printer 424
13.23 Monitoring Network Devices 426
13.24 Monitoring VMware Machines 429
13.25 Monitoring Windows witn NSClient++ 432
13.26 Monitoring Windows Devices 434

14 How to contribute 439
14.1 Shinken packs 440
14.2 Shinken modules and Shinken packs 442
14.3 Help the Shinken project 444
14.4 Getting Help and Ways to Contribute 444
14.5 Shinken Package Manager 445

15 Development 449
15.1 Shinken Programming Guidelines 450
15.2 Test Driven Development 452
15.3 Nagios Plugin API 455

15.4 Developing Shinken Daemon Modules . . . 458
15.5 Hacking the Shinken Code . . . 458

16 Shinken modules **463**
16.1 Shinken modules . . . 464
16.2 Amazon AWS/EC2 import . . . 13232
16.3 Amazon AWS/EC2 import . . . 13234
16.4 The distributed retention modules . . . 13235
16.5 How to enable and use Livestatus module . . . 13238
16.6 Exporting data for reporting . . . 13239
16.7 Monitoring Linux System with Glances and checkglances.py . . . 13240
16.8 Shinken GLPI integration . . . 13241
16.9 Ip Tag module . . . 13242
16.10 Ubuntu Landscape import . . . 13243
16.11 Shinken Livestatus API . . . 13245
16.12 NSCA module . . . 13246
16.13 Retention troubleshooting . . . 13247
16.14 NRPE Module . . . 13248
16.15 Extending Shinken . . . 13249
16.16 Broker modules . . . 13250
16.17 TSCA (Thrift Service Check Acceptor) . . . 13257
16.18 VMWare Arbiter module . . . 13259
16.19 Web Service Module . . . 13260
16.20 SNMP module . . . 13261
16.21 WebUI module . . . 13277

17 Reference **13281**
17.1 shinken . . . 13281
17.2 shinken Package . . . 13336
17.3 clients Package . . . 13392
17.4 daemons Package . . . 13394
17.5 discovery Package . . . 13394
17.6 misc Package . . . 13394
17.7 objects Package . . . 13397
17.8 webui Package . . . 13429

18 Indices and tables **13431**

Python Module Index **13433**

Contents:

CHAPTER 1

About

1.1 About Shinken

1.1.1 What Is It?

Shinken is a system and network monitoring application. It supervises hosts and services from an IT and business point of view. Alerting or taking action on failures and recovery.

Shinken is a monitoring tool compatible with Nagios configuration, plugins and interfaces. It is written in Python, so it should work under all Python supported platforms.

Some of the many features of Shinken include:

- Web 2.0 Interface named WebUI that has innovative methods to visualize the state of your systems
- Web 2.0 Interface named SkonfUI, that links discovery and configuration management
- Livestatus networked API to provide realtime access to performance, status and configuration data * Interchangeable console or mobile interfaces
- Providing operational or business insight * Ability to map business processes to Hosts and services * Ability to associate a business impact metrics to any warning or outages of hosts and services used to deliver a business process * Ability to define network host hierarchy using "parent" hosts, allowing detection of and distinction between hosts that are down and those that are unreachable
- Monitoring Hosts and Services * Simple plugin design that allows users to easily develop their own service checks * Monitoring of network services ("SMTP", "POP3", "HTTP", "NTP", PING, etc.) * Monitoring of host resources (processor load, disk usage, etc.) * Hundreds of Nagios check scripts to choose from * High performance plugin modules integrated in the distributed daemons to easily extend the scope of the software * Parallelized service and host checks * Designed for highly available and load balanced monitoring * Acquire performance data from collectd via its network interface
- Define Triggers in the Shinken core to calculate new performance metrics or states based on performance or state data
- Contact notifications when service or host problems occur and get resolved (via email, SMS, pager, or user-defined method)
- Ability to define event handlers to be run during service or host events for proactive problem resolution
- Integrates with PNP4Nagios and Graphite time-series databases for storing data, querying or displaying data.
- Supports distributed retention modules, caches and databases to meet persistence and performance expectations

1.1.2 System Requirements

The *requirement for running Shinken* are the Python interpreter and a very short list of Python modules. You will also want to have TCP/IP configured, as the typical installation will download packages to install from the Internet and most service checks will be performed over the network.

1.1.3 Legacy Requirements

You are not required to use the original Nagios user interface CGIs included with Shinken. However, if you do decide to use them for migration purposes, you will need to have the following software installed...

- A web server (preferrably Apache)
- Thomas Boutell's gd library version 1.6.3 or higher.

1.1.4 Licensing

Shinken is licensed under the terms of the GNU Affero General Public License as published by the Free Software Foundation. This gives you legal permission to copy, distribute and/or modify Shinken under certain conditions. Read the 'LICENSE' file in the Shinken distribution or read the online version of the license for more details.

Shinken is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

The Shinken documentation is based on Nagios so is licensed under the terms of the GNU General Public License Version 2 as published by the Free Software Foundation. This gives you legal permission to copy, distribute and/or modify Shinken under certain conditions. Read the 'LICENSE' file in the Shinken distribution or read the online version of the license for more details.

Shinken is provided AS IS with NO WARRANTY OF ANY KIND, INCLUDING THE WARRANTY OF DESIGN, MERCHANTABILITY, AND FITNESS FOR A PARTICULAR PURPOSE.

1.1.5 Acknowledgements

A long list of people have contributed to Shinken. The THANKS file included as part of Shinken and the project page at http://www.shinken-monitoring.org provide more details.

1.1.6 Downloading The Latest Version

You can check for new versions of Shinken at http://www.shinken-monitoring.org.

1.2 Feature comparison between Shinken and Nagios

Shinken is not just a fork of Nagios. It has been rewritten completely as a modern distributed application while maintaining compatibility with the Nagios configuration, LiveStatus API and check plugins.

The major differences are listed on the page http://www.shinken-monitoring.org/what-is-in-shinken-not-in-nagios-and-vice-versa/.

1.2.1 Change Log

The **Changelog** file is included in the source root directory of the source code distribution.

1.3 Shinken notable innovations

1.3.1 Summary

- Shinken is a true distributed applications which splits the different roles into separate daemons
- Shinken permits the use of modules to extend and enrich the various Shinken daemons
- Shinken is 100% python, from its API, frontend, back-end, discovery engine and high performance poller modules

- Scalability * Shinken has a powerful scheduler for supervising tens of thousands of devices v1.2 * Shinken can supervise multiple independent environments/customers * Shinken uses MongoDB to provide a distributed and highly scalable back-end for storing historical event data (experimental)
- Graphical and statistical analysis * Shinken provides integration with the modern time series database, Graphite * Shinken provides triggers against performance or event data in the core and in external checks (experimental) v1.2
- Correlation * Shinken differentiates the business impact of a critical alert on a toaster versus the web store * Shinken supports efficient correlation between parent-child relationship and business process rules

1.3.2 Notable items for DevOps admins

- Use Shinken and Graphite seamlessly in the Shinken WebUI. v1.2
- Export data from Shinken to Graphite and manipulate the data point names with PRE, POST, and SOURCE variables
- Buffered output to Graphite.
- Multiple Graphite destination servers.
- Use check_graphite to poll graphite and generate alerts.
- Use wildcards in checks against Graphite.
- Auto-detection, logging and semi-automatic registration of new passive checks. Planned for v1.4
- Mix and match frontends(Multisite, Nagvis), plugins, alerting(Pagerduty), analysis (Splunk, Logstash, Elastic Search, Kibana)

1.3.3 Notable items for Nagios admins

- Modern, Scalable, HA and distributed out of the box.
- Business rules, business impacts levels integrated in the core.
- The code is approachable for sys admins to improve and customize the core and manage additions using modules.
- Supervision packs(templates+commands+etc) and community
- For a full comparison: Shinken versus Nagios page.
- Can you say Graphite integration..

Shinken *is* the modern Nagios, re-implemented in Python, end of story.

1.3.4 Notable items for Zabbix admins

- A powerful and efficient dependency model that does event suppression. Not as flexible as the great Zabbix calculated items, but suffers from much less false positives, which is the whole point, especially at 3am.
- Uses the Graphite time-series database, which is hands-down one of the most modern, easy to use, fast, powerful and flexible time-series databases. None of the slooowness associated with scaling a SQL based storage for time-series.
- The code is approachable for sys admins to improve and customize the core and using modules.

- The new SNMP poller is more intelligent and efficient in its collection against remote devices. Zabbix collects each metric as it is scheduled, so IN/OUT stats of the same interface could be collected at two different times, errr, say what!

1.3.5 Notable items for Zenoss admins

- A working dependency model that does good and efficient event suppression and correlation.
- 100% open-source...
- Can you say Graphite integration, MongoDB, distributed architecture and seamless HA.

1.4 The project Vision

1.4.1 What we want and how we want it

We are here to create an open source monitoring solution that is:

- easy to install;
- easy to use for new users;
- useful for sysadmins and business types;
- built for scalability;
- incorporates high availability;
- multi-platform (and not just Unix ones);
- utf8 compliant;
- nearly fully compatible with Nagios configuration;
- fully compatible with its plugins and interfaces;
- fast enough
- centralized configuration that should be as easy as possible for the admin to manage;
- independent from any huge monitoring solution;
- fun to code :)

More precisely:

- The 2 first points are to make the system accessible to more people by automating and pre-building the typical monitoring environments so that users can focus on customizing, not just getting the basics working.
- Shinken provides a business monitoring service, not just an IT focused service. This means helping users to differentiate between real business impacting outages and minor IT related failures.
- The next two are for the global architecture. We should be able to scale the load within minutes without losing the high availability feature. We believe this has been successfully implemented. :)
- Multi platform is also important. If we only focused on GNU/Linux, a large user segment would be left out (Windows!). Of course, some features inherited from Nagios are Unix-only like named pipes, but we should promote other ways for users to bypass this (like commands sent to livestatus or in a web based API for example). Such "os limited" points must be *limited to modules*.

This is what we want to have (and already have for the most part).

1.4.2 What we do not want

At the beginning of the project, we used to say that what we didn't want was our own webui. This became a requirement. Why? Because none of the current UIs truly provided the right way to prioritize and display business oriented information to the end users. Features like root problem analysis or business impact. After trying to adapt the external UIs, we had to roll our own. So we did. :)

So there is nothing we do not want, if in the end it helps Shinken's users.

1.5 Feature selection and release cycle

1.5.1 Feature selection

The Shinken Ideascale is a key input in the feature selection process, everyone can propose and vote for his favorite feature. Features are now managed based on expected user value. Features that users desire the most and which fit the project vision will be prioritized. The feature vote should be promoted to all users. The more users vote, the more we are sure to give them the most value for their monitoring system.

1.5.2 Release cycle

- "value first priority"
- Major changes are handled in github forks by each developer
- Very open to user submitted patches
- Comprehensive automated QA to enable a fast release cycle without sacrificing stability
- Tagging experimental unfinished features in the documentation
- Release soon and release often mentality

1.6 Release code names

I (Jean Gabès) keep the right to name the code name of each release. That's the only thing I will keep for me in this project as its founder. :)

CHAPTER 2

Getting Started

2.1 Advice for Beginners

Congratulations on choosing Shinken! Shinken is quite powerful and flexible, but it can take a bit of work to get it configured just the way you'd like. Once you become familiar with how it works and what it can do for you, you'll never want to be without it :-) Here are some important things to keep in mind for first-time Shinken users:

- **Relax - it takes some time.** Don't expect to be able to get things working exactly the way you want them right off the bat. Setting up Shinken can involve a bit of work - partly because of the options that Shinken offers, partly because you need to know what to monitor on your network (and how best to do it).
- **Use the quickstart instructions.** The *Quickstart installation guide* is designed to get most new users up and running with a basic Shinken setup fairly quickly. Within 10 minutes you can have Shinken installed and monitoring your local system. Once that's complete, you can move on to learning how to configure Shinken to do more.
- **Get familiar with the Getting Started section.** Shinken has a getting started documentation section, that is easier to read through for common undertakings and to understand how Shinken works. It can be simpler to follow than the exhaustive official documentation.
- **Read the documentation.** Shinken can be tricky to configure when you've got a good grasp of what's going on, and nearly impossible if you don't. Make sure you read the documentation (particularly the sections on "Configuring Shinken" and "The Basics"). Save the advanced topics for when you've got a good understanding of the basics.
- **Seek the help of others.** If you've read the documentation, reviewed the sample config files, and are still having problems, go through the *Shinken user resources* to learn how you can get help and contribute back to the project.

2.2 Quickstart Installation Guides

Abstract

These quickstart guides are intended to provide you with simple instructions on how to install Shinken and have it monitoring your local machine inside of 10 minutes. No advanced installation options are discussed here - just the basics that will work for 95% of users who want to get started.

2.2.1 Guides

Installation guides are currently available for the following OSs :

- *10 Minute installation Guide for GNU/Linux and Windows*
- *Shinken installation requirements*
- *Shinken 1.2 installation requirements*
- *Shinken installation script*
- *GNU/Linux Quickstart*
- *Windows Quickstart*
- *Nokia N900*
- *Shinken/Thruk/PNP4Nagios on Red Hat*

2.2.2 Post-Installation Modifications

Once you get Shinken installed and running properly, you'll no doubt want to start monitoring more than just your local machine. Check out the following docs for how to go about monitoring other things:

- *Monitoring Windows machines*
- *Monitoring GNU/Linux or other Unix machines*
- Monitoring Netware servers
- *Monitoring routers/switches*
- *Monitoring network printers*
- *Monitoring publicly available services ("HTTP", "FTP", "SSH", etc.)*

Also, you can check the next documentations:

- How to monitor

2.3 Installations

2.3.1 Shinken Requirements

Shinken provides an "install script" which tries to manage all necessary steps to install and get Shinken up and running. Use it if your Operating System is compatible with it, otherwise use the Fedora RPMs or the *setup.py method*.

Get me back to the *10 minute installation guide*.

Mandatory Requirements

Shinken installation prefix:

You must use the same installation prefix on ALL your Shinken hosts.

Shinken requires on all hosts running a Shinken daemon the *SAME* versions :

- Python 2.4 or higher
  - Python 2.6 or higher is mandatory on the server running the built-in Web interface (WebUI)
  - Python 2.6 or higher is mandatory for using the discovery engine and configuration web interface (SkonfUI)
- setuptools or the newer'distribute'_
- pyro Python package
  - version 3.x for Debian Squeeze
  - version < 4.14 if running Shinken 1.0.1
  - version 3.x, 4.x and if you can 4.15 for Shinken 1.2 and newer
- multiprocessing Needed when using Python 2.4 or 2.5 (already included in Python 2.6 and higher)
-
- python-dev Python package or distribution package (ex. python-dev under Ubuntu)Core, Libs and Modules
- build-essentials
  - sudo apt-get install build-essential python-dev (Installation under Linux Ubuntu/Debian)
- pymongo >= 2.1 : for WebUI

Conditional Requirements

If you plan on using the *Livestatus module*, or a third party web interface, you will also need at a minimum the following Python packages:

- simplejson (Included in Python 2.6 and higher)
- pysqlite

If you plan on using SkonfUI, NPCMOD or Canopsis

- pycurl for SkonfUI configuration pack management
- kombu for the Canopsis broker Module
- MySQL_python:MySQLdb >= <version> for ndomod

If you plan on developing or testing features, you will also need at a minimum the following Python packages:

- nose

- unittest
- git-core If you want to regularly checkout the latest code and contribute to the project. Though you can also simply download the latest from the github shinken website

if you plan on using check scripts installed by install.sh

- paramiko (only if you use installer script)
- netifaces (only if you use installer script)

Installing/Checking Common Requirements on Linux

Python

For Python itself, the version which comes with almost all distributions should be okay. Though, if you are using a distribution with Python 2.4 or 2.5, **you should use a Python version of least 2.6 in a virtualenv**. This will avoid problems for upcoming Shinken versions that will require Python 2.6 and higher. (*This is aimed right at you, RHEL 5!*)

You must use the same version of Python on ALL your Shinken hosts.

You can validate your Python version with:

```
python -c 'import sys; print sys.version[:3]'
```

Pyro

You can validate your installed Python Pyro module version using:

```
python -c 'try:; import Pyro; except ImportError:; import Pyro4 as Pyro; print Pyro.constants.VERSION
```

You must use the same version of Pyro on ALL your Shinken hosts.

Under Ubuntu, you can grab the Pyro module with:

```
sudo apt-get install pyro
```

Under other distributions, you can search for it:

```
yum search pyro
```

If you do not find it, or need to install a specific version, you can install it from PyPI:

```
easy_install pyro
```

If you do not find it, or need to install a specific version, you can install it from PyPI using the following Syntax:

```
pip install pyro4-4.15
```

2.3.2 Prerequisites for Shinken 1.2

This page lists the prerequisites of shinken. A packager can use these dependencies when creating a package for his distribution and others can use it as a reference when installing on a new OS which is unsupported until now.

Build Dependencies for certain modules

These packages are only needed if one or more of the required modules are not contained in the distros repositories or only in a version that is not sufficiently high enough.

The package names are from Debian/Ubuntu, please adapt as needed

- build-essential
- libperl-dev
- libsqlite3-dev
- python-dev
- libmysqlclient-dev
- libevent-dev
- python-setuptools

Core, Libs and Modules

- Python >= 2.4 for CORE
- Python >= 2.6 for WebUI and Skonf
- Pyro >= 4.0, if possible 4.9 or 4.14
- pysqlite:sqlite3 >= <version> for livestatus
- MySQL_python:MySQLdb >= <version> for ndomod
- pymongo >= 2.1 : for WebUI
- pycurl for Skonf configuration pack management
- paramiko (only if you use installer script)
- netifaces (only if you use installer script)

Optional Python Modules for Extended Features

- kombu : only if you use canopsis module
- simplejson : only if python 2.5 used
- python-ldap : active directory authentication (needed by Shinken WebUI ActiveDir_UI module)
- ujson : ujson (preferred by Livestatus over simplejson)
- Python >= 2.7 for running the Shinken Test Suite. **This is mandatory for developers.**

Additional Software

- sqlite3
- nmap : for discovery feature
- unzip
- nagios-plugins
- mongodb

2.3.3 10 Minute Shinken Installation Guide

Summary

By following this tutorial, in 10 minutes you will have the core monitoring system for your network.

The very first step is to verify that your server meets the requirements, the installation script will try to meet all requirements automatically.

You can get familiar with the *Shinken Architecture* now, or after the installation. This will explain the software components and how they fit together.

- Installation : *GNU/Linux & Unix*
- Installation : Windows
- Post-Installation : Common

Ready? Let's go!

GNU/Linux & Unix Installation

Method 1: Installation Script

> **Warning:** Do not mix installation methods! If you wish to change method, use the uninstaller from the chosen method THEN install using the alternate method.

The `install` script is located at the root of the Shinken sources. It creates the user and group, installs all dependencies and then installs Shinken. It is compatible with Debian, Ubuntu, Centos/RedHat 5.x and 6.x. The only requirement is an internet connection for the server on which you want to install Shinken.

Basic automated installation You can get the sources and launch the install script with just this command :

```
curl -L http://install.shinken-monitoring.org | /bin/bash
```

You can then jump to the "*Start Shinken*" section and continue from there.

If instead want to make it manually, go in the next step :)

Installation using the sources Download stable Shinken v1.4 tarball archive (or get the latest git snapshot) and extract it somewhere:

```
cd ~
wget http://www.shinken-monitoring.org/pub/shinken-1.4.tar.gz
tar -xvzf shinken-1.4.tar.gz
```

By default the installation path is `/usr/local/shinken`, but it can be specified in the configuration file (see install.d/README).

Run a basic installation *You need to have lsb-release package installed.*

```
cd ~/shinken-1.4
./install -i
```

Done! Shinken is installed and you can edit its configuration files in `/usr/local/shinken/etc` (by default).

Init.d scripts are also copied, so you just have to enable them at boot time (with `chkconfig` or `update-rc.d`).

Start Shinken To start Shinken:

```
/etc/init.d/shinken start
```

But wait! The installation script can do much more for you, such as installing plugins, addons, upgrading and removing an installation. See the install.d/README file or *full install script doc* to know how you can get the best out of it.

Run a full installation To list the plugins and addons available:

```
./install -h
```

A common and fully featured installation is:

```
./install -i &&\
./install -p nagios-plugins &&\
./install -p check_mem &&\
./install -p manubulon &&\
./install -a multisite &&\
./install -a pnp4nagios &&\
./install -a nagvis &&\
./install -a mongodb
```

This will automatically install:

- Shinken
- Nagios plugins
- Manubulon SNMP plugins
- Multisite
- PNP4Nagios
- Nagvis
- MongoDB # This is used for the SkonfUI beta and WebUI

Tip: If you encounter problems installing Multisite, it may be because the latest stable version on Check_MK's website has changed. Simply change the MK version in `install.d/shinken.conf` to the latest stable version: `export MKVER="1.2.0p3"`

For more information regarding the install script. See the *full install script doc*

Update See *update Shinken*.

Method 2: On Fedora with RPM

Warning: Shinken is an official RPM

First install Python Pyro

```
yum install python-pyro
```

Then install Shinken

```
yum install shinken  shinken-poller\
shinken-scheduler shinken-arbiter \
shinken-reactionner shinken-broker shinken-receiver
```

Enable Shinken services

```
for i in arbiter poller reactionner scheduler broker; do
systemctl enable shinken-$i.service;
done
```

Start Shinken services

```
for i in arbiter poller reactionner scheduler broker; do
systemctl start shinken-$i.service;
done
```

Stop Shinken services

```
for i in arbiter poller reactionner scheduler broker; do
systemctl stop shinken-$i.service;
done
```

Easy is not it?

Windows Installation

Method 1: Packaged .EXE Installer

Download the Executable installer Download the executable installer for Shinken 1.4.

Thanks to J-F BUTKIEWICZ for preparing the installation package.

Read the installation instructions Installation instructions at http://www.veosoft.net/index.php/en/tutorials/shinken-tutorials/shinken-1-2-4-installation-on-windows-2008-r2

Run the installer What? You don't want to read them? No problem.

Simply launch the .exe and click Next until the installation has run its course. :-)

The executable installer creates service and copies the necessary files into C:/Program Files (x86)/Shinken by default, but you can change that target folder as you want.

CHECK_WMI_PLUS configuration By default, check_wmi_plus.pl use an user/password to access the windows WMI functions. But locally (shinken host managed itself on windows), this cannot be done. So the local template always works even if a wrong user/password is set. In the commands file, just set local to user and localhost to the computer.

But now, how to configure shinken to manage others windows hosts using wmi. Shinken team provides a set of commands in the windows template. We will see how to set the user/password to work properly. But there is an

"extra" function to use the poller's service to push its credential to check_wmi_plus. This kind of configuration and globaly the use of check_wmi_plus under windows is described in this *link*.

Post installation

Where is the configuration?

The configuration is where you put the `etc` directory. Usually it's `/etc/shinken`, `/usr/local/shinken/etc` or `C:/Pro`

- `nagios.cfg` is meant to be fully compatible with Nagios;
- `shinken-specific.cfg` contains all Shinken specific objects (ie. daemons, realms, etc.).

Do I need to change my Nagios configuration?

No, there is no need to change your existing Nagios configuration. You can use an existing Nagios configuration as-is, as long as you have installed the plugins expected by the configuration.

Once you are comfortable with Shinken you can start to use its unique and powerful features.

What do I need to do next

The next logical steps for a new user are as listed in the Getting Started page:

- Setup the web interface:
  - Use the *default WebUI* (Note: it's the mandatory interface on Windows)
  - Or set-up a *third-party web interface* and addons.
- Did you read the *Shinken Architecture* presentation?
- Complete the *Shinken basic installation*
- Start adding devices to monitor, such as:
  - *Public services* (HTTP, SMTP, IMAP, SSH, etc.)
  - *GNU/Linux* clients
  - *Windows* clients
  - *Routers*
  - *Printers*

Getting Help

New and experienced users sometimes need to find documentation, troubleshooting tips, a place to chat, etc. The *Shinken community provides many resources to help you*. You can discuss installation documentation changes in the Shinken forums.

2.3.4 Review of script's option and parameters

Important: It's recommended to use option one by one! Some options have a parsing trick that prevent from being mixed with other. Call the install script as much as you need is better.

Here are the options for the install script :

`-k | --kill`	Kill shinken
`-i | --install`	Install shinken
`-u | --uninstall`	Remove shinken and all of the addons. if an argument is specified then just remove the specified argument
`-b | --backup`	Backup shinken configuration plugins and data
`-r | --restore`	Restore shinken configuration plugins and data
`-l | --listbackups`	List shinken backups
`-c | --compresslogs`	Compress rotated logs
`-e | --enabledaemons`	Which daemons to keep enabled at boot time. It's a quoted list of shinken daemons that should be enabled at startup. Daemons are: * arbiter * scheduler * poller * broker * reactionner * receiver * skonf
`-p | --plugin`	Install plugins. Argument should be one of the following: * check_esx3 * nagios-plugins * check_oracle_health * check_mysql_health * capture_plugin * check_wmi_plus * check_mongodb * check_emc_clariion * check_nwc_health * manubulon (snmp plugins) * check_hpasm * check_netapp2 * check_mem (local enhanced memory check plugin) * check_snmp_bandwidth (check bandwidth usage with snmp) * check_netint (enhanced version of check_snmp_int plugins) * check_IBM * check_IBM_DS * check_HPUX * check_rsync
`-a | --addon`	Install addons. Argument should be one of the following: * pnp4nagios * multisite * nagvis * mongodb * nconf (experimental) * notify_by_xmpp
`--nconf-import`	Import shinken packs into nconf (experimental). This require nconf to be installed
`--register`	Register on community (experimental)
`--enableeue`	Enable application performance monitoring stack (experimental)
`-h | --help`	Show help

2.3.5 Review of variable used in the script

The following variables can bet set before running the script in order to modify its behavior.

Unordered List ItemTMP	Specify the temp folder for Shinken installation. Default : /tmp
LOGFILE	Specify the logs file used during Shinken installation. Default : $TMP/shinken.install.log"
USEPROXY	Specify if the server is behind a proxy. Used during installing prerequisites. Default : 0 (No)
SKIPPREREQ-UISITES	Specify if the script check and install Shinken prerequisites. All prerequisites are not compulsory for a "small" install Default : 0 (No)
TARGET	Specify the target directory for Shinken installation. Default : /usr/local/shinken
ETC	Specify the etc directory for Shinken installation (configuration files directory). Default : $TARGET/etc
LIBEXEC	Specify the libexec directory for Shinken installation (plugin directory). Default : $TARGET/libexec
VAR	Specify the var directory for Shinken installation(logs directory). Default : $TARGET/var
BACKUPDIR	Specify the backup directory. Default : /opt/backup
SKUSER	Specify the Shinken user. Default : shinken
SKGROUP	Specify the Shinken group. Default : shinken
MANAGEPYRO	Specify if the script will install a "managed" version of Pyro (Pyro4 lib) Defaukt : 0 (No)
KEEP-DAYSLOG	Specify the number of days to keep the logs (for Sqlite purge). Default : 7
RETENTION-MODULE	Specify the default retention module (pickle|mongodb). Default pickle

2.3.6 Alternatives installations

Here you can find another way to install Shinken without using shinken.sh . If you want to understand deeply how Shinken work, it may be a better solution.

Note: Article under creation, it will me deeply modified before the 1.0 release ;)

Installation

On Ubuntu or Debian

Save time, use the very good installation script from Nicolargo! Thanks a lot to him :)

```
cd ~
wget https://raw.github.com/nicolargo/shinkenautoinstall/master/shinkenautoinstall-debian.sh
chmod a+x shinkenautoinstall-debian.sh
sudo ./shinkenautoinstall-debian.sh
```

On windows

Get the http://shinken-monitoring.org/pub/shinken-0.8.1.tar.gz file in c:shinken

During portions of the installation you'll need to have administrator access to your server. Make sure you've installed the following packages on your Windows installation before continuing.

- Python 2.7 for Windows
- Pyro 4 library Windows
- Windows Resource Kit
- PyWin32

Take the files instsrv.exe and srvany.exe from the directory of the resource kit (typically "c:program filesWindows Resource KitsTools") and put them in the directory "c:shinkenwindows" (it should already exist by decompressing the archive, or you are a directory level to deep).

To install all services, launch the installation batch file:

```
c:\shinken\windows\install-all.bat
```

Launch services.msc to see you brand new services (Shinken-*).

But don't start them yet, you should jump up to the discovery part before starting new Shinken services.

On Fedora with RPM

Prerequisites:

Install python Pyro:

```
# yum install python-pyro
```

Install Shinken:

Download RPM to the url http://hvad.fedorapeople.org/fedora/shinken/RPM/

```
# yum localinstall --nogpgcheck shinken-0.8.1-1.fc15.noarch.rpm shinken-arbiter-0.8.1-1.fc15.noarch.r
```

Enable Shinken services:

```
# for i in arbiter poller reactionner scheduler broker; do systemctl enable shinken-$i.service ; done
```

Start Shinken services:

```
# for i in arbiter poller reactionner scheduler broker; do systemctl start shinken-$i.service ; done
```

Stop Shinken services:

```
# for i in arbiter poller reactionner scheduler broker; do systemctl stop shinken-$i.service ; done
```

On Debian with DEB packages

Shinken is packaged on the debian "sid": Prerequisites:

```
aptitude install shinken python-simplejson python-pysqlite2 python-mysqldb python-redis python-memcac
```

Start Shinken services:

```
# for i in  broker poller reactionner receiver scheduler arbiter ; do /etc/init.d/shinken-$i start ;d
```

Stop Shinken services:

```
for i in  broker poller reactionner receiver scheduler arbiter ; do /etc/init.d/shinken-$i stop ;done
```

On RedHat/Centos and other GNU/Linux box: from the sources

Shinken asks for few dependencies:

- Python >= 2.4 (but Python 2.6 is preferable)
- Pyro, a Python module (version >= 4.5 is possible)

To know which Python version you are running, just type

```
python -V
```

Dependencies for Debian folks To get Dependencies launch:

```
sudo apt-get install pyro nagios-plugins-extra
```

Dependencies for Centos5/RH5 with python 2.4

Important: Python version 2.4 is the default version of python on CentOS/RH5, so this is the easiest way to install Shinken on CentOS. The problem is that some advanced Shinken functionalities need ''at least" python 2.6

First get the dependencies (as root or with sudo):

```
yum install gcc nagios-plugins python-devel python-simplejson
wget http://pypi.python.org/packages/source/P/Pyro/Pyro-3.10.tar.gz#md5=7fc6b8b939073d4adb0e8939c59aa
tar xvfz Pyro-3.10.tar.gz
cd Pyro-3.10
python setup.py install

cd ~

wget http://pypi.python.org/packages/2.4/s/setuptools/setuptools-0.6c11-py2.4.egg#md5=bd639f9b0eac4c4
sh setuptools-0.6c11-py2.4.egg

cd ~

wget http://pypi.python.org/packages/source/m/multiprocessing/multiprocessing-2.6.2.1.tar.gz#md5=5cc4
tar xvfz multiprocessing-2.6.2.1.tar.gz
cd multiprocessing-2.6.2.1/
python setup.py install
```

Dependencies for Centos5/RH5 with python 2.6

Important: Python version 2.4 is the default version on CentOS5/RH5. This version of python is deeply linked to the OS (yum package manager for instance), so you can't just ''update" python. In order to add python 2.6 on your system, you will need to add packages from at least 2 additional repositories: ''RPMForge'' and ''EPEL release''

First, add the additional repositories

```
wget http://apt.sw.be/redhat/el5/en/i386/rpmforge/RPMS/rpmforge-release-0.5.2-2.el5.rf.i386.rpm
rpm -Uvh rpmforge-release-0.5.2-2.el5.rf.i386.rpm
wget http://download.fedora.redhat.com/pub/epel/5/i386/epel-release-5-4.noarch.rpm
rpm -Uvh epel-release-5-4.noarch.rpm
```

Then install the dependencies

```
yum install gcc nagios-plugins python26-devel python26-simplejson
wget http://pypi.python.org/packages/source/P/Pyro4/Pyro4-4.11.tar.gz#md5=8126e7049206b7b09f324750f50
tar xvfz Pyro4-4.11.tar.gz
cd Pyro4-4.11
python26 setup.py install

cd ~
wget http://pypi.python.org/packages/2.6/s/setuptools/setuptools-0.6c11-py2.6.egg#md5=bfa92100bd772d5
sh setuptools-0.6c11-py2.6.egg
```

Shinken installation

Create the Shinken user:

```
sudo adduser shinken
sudo passwd shinken
```

Important: Be sure to create a valid home directory for the shinken user. If not, the daemons won't start.

Then, get Shinken package and install it:

```
cd ~
wget http://www.shinken-monitoring.org/pub/shinken-0.8.1.tar.gz
tar xfz shinken-0.8.1.tar.gz
cd shinken-0.8.1
sudo python setup.py install --install-scripts=/usr/bin
sudo mkdir -p /usr/lib/nagios/plugins/
sudo cp libexec/* /usr/lib/nagios/plugins/
```

Important: Replace ''python26" instead of ''python" in the command line "sudo python setup.py install –install-scripts=/usr/bin" if you run CentOS5/RH5 and you wish shinken to be installed with python 2.6 support

You will get:

```
new binaries into /usr/bin (files shinken-*)
some new checks and notification scripts in /usr/lib/nagios/plugins/
some new directory (/etc/shinken and /var/lib/shinken).
```

Discover your network

The network discovery scans your network and sets up a standardized monitoring configuration for all your hosts and network services. To run it, ou need to install the nmap network discovery tool.

Ubuntu:

```
sudo apt-get install nmap
```

RedHat/Centos:

```
yum install nmap
```

Now, you are ready to run the network discovery tool:

```
[ -d /etc/shinken/discovery ] && sudo mkdir /etc/shinken/discovery
sudo shinken-discovery -o /etc/shinken/discovery -r nmap -m "NMAPTARGETS=192.168.0.1-254 localhost"
```

The important part is the NMAPTARGETS value. It's an nmap target value, so you can give the value you want, like a list of hosts or an IP range.

Note: The scan duration depends on the number of IP addresses to scan. If you are scanning a large network, the scan can run into the tens of minutes. (the scan timeout is set to one hour by default. The timeout parameter is defined in the etc/discovery.cfg)

Setup Thruk, the Web interface

This next section will use the Nicolargo installation script, thank you once again.

If you have already run the Nicolargo script for Debian, you can skip this part.

To install Thruk, launch:

```
perl -V:version -V:archname
browse to http://www.thruk.org/files/  and download the file that matches the archname and version
( version is last number before tar.gz)
tar zxvf *filename*
cd Thruk-$thruk_version
wget https://raw.github.com/nicolargo/shinkenautoinstall/master/thruk_local.conf
cd ..
cp -R Thruk-1.0.5 /opt/thruk
chown -R shinken:shinken /opt/thruk
wget -O /etc/init.d/thruk https://raw.github.com/nicolargo/shinkenautoinstall/master/thruk
chown root:root /etc/init.d/thruk
chmod a+rx /etc/init.d/thruk
```

For Ubuntu/Debian:

```
update-rc.d thruk defaults
```

For RedHat/Centos:

```
chkconfig thruk --add
```

First launch

You are now ready to start the system, launch Shinken and Thruk.

```
/etc/init.d/shinken start
/etc/init.d/thruk start
```

You can validate that the software is running smoothly by *tailing* the main log file at:

```
tail -f /var/lib/shinken/shinken.log
```

And by connecting to the web interface at http://localhost:3000 (or use the IP address of your server)

Congrats, you just launched your next monitoring tool ^_^

Now you can go through the rest of the wiki to learn how to work with the configuration, and customize it as you need. There are tutorials in the getting started section for common tasks and there is an official documentation manual that provides in depth coverage of features and options.

Now are ready to learn how to configure the Shinken daemons, your gentle introduction to distributed monitoring, by reading the *configure Shinken* page.

2.3.7 Shinken on RedHat 6 with Thruk and PNP4Nagios HOWTO

We'll install Shinken with the Thruk web user interface and the PNP4Nagios graphs. We'll also configure SNMP, NRPE and SSH access to the monitored hosts.

Packages will be installed in /opt unless they are already packaged.

Shinken

Prepare sources:

```
mkdir /opt/shinken-dl/
wget http://www.shinken-monitoring.org/pub/shinken-1.4.tar.gz
tar xzf shinken-1.4.tar.gz
```

Installation:

```
cd /opt/shinken-dl/shinken-1.4/
TARGET=/opt/shinken SKUSER=shinken SKGROUP=shinken ./install -i
./install -p nagios-plugins
./install -p manubulon  # snmp checks
./install -p check_netint  # network/trafic checks
```

Mandatory configuration:

- In `/opt/shinken/etc/shinken-specific.cfg`, define `auth_secret` with a random password
- Fix path to the `mail` command:

```
sed -i -e 's,/usr/bin/mail,mail,' /opt/shinken/etc/commands.cfg
```

Optional configuration:

- Change in `/opt/shinken/etc/nagios.cfg`:
- Avoid flapping due to having the same timeout for service checks (UNKNOWN) and for check_https (CRITICAL):

```
service_check_timeout=60
```

- Support long event handlers:

```
event_handler_timeout=300
```

- Change in templates.cfg:
- If you need hosts that can't be ping'd, comment out in `generic-host`:

```
#check_command                  check_host_alive
```

- Notifications may be sent even if the host is out of its notification hours, but you can force host>service inheritance by commenting this in `generic-service`:

```
#notification_period             24x7
```

- Same for check periods:

```
#check_period             24x7
```

- Notifications are sent every hour by default, you can change that to every day:

```
notification_interval           1440
```

- Add 'u,f' to *service notifications* in `notificationway:service_notification_options`
- If you need a global event handler (workaround issue 717), modify `generic-service`:

```
event_handler_enabled           1
event_handler                   test_log_service
```

Mail

In case you need to configure the Shinken mail sender:

```
echo "shinken   shinken-notifications@mydomain.tld" >> /etc/postfix/canonical
postmap /etc/postfix/canonical
cat <<'EOF' >> /etc/postfix/main.cf
sender_canonical_maps = hash:/etc/postfix/canonical
EOF
```

Shinken also sends mail to none@localhost which is the contact for user 'guest'. This triggers bounces, so you can auto-trash these mails:

```
echo 'none: /dev/null' >> /etc/aliases && newaliases
```

Thruk

Follow *use_with_thruk*:

- Installation from RPM (http://www.thruk.org/download.html):

```
rpm -ivh http://www.thruk.org/files/pkg/v1.76-3/rhel6/x86_64/thruk-1.76-3.rhel6.x86_64.rpm
```

- *SELinux configuration* (or disable it with `setenforce Permissive`)
- *Using Shinken with Thruk*

Thruk is available at: http://YOUR_SHINKEN_IP/thruk/

PNP4Nagios

Follow *use_with_pnp*:

- Go to the Shinken sources and set the installation path in `/opt/shinken-dl/shinken-1.4/install.d/shinken`

```
PNPPREFIX=/opt/pnp4nagios
```

- *Install PNP4Nagios automatically*
- *Using Shinken with PNP4Nagios*

PNP4Nagios is now linked from Thruk though `action_url`, and more generally available at http://YOUR_SHINKEN_IP/pnp4nagios/

Monitored hosts

SNMP

Let's enable SNMP on our monitored hosts.

```
# Install SNMP server:
yum install net-snmp

# Read-only access:
echo "rocommunity public" > /etc/snmp/snmpd.conf

# Don't log each SNMP request:
[ -e /etc/sysconfig/snmpd ]         && echo 'OPTIONS="-LS0-4d -Lf /dev/null -p /var/run/snmpd.pid"'
```

```
[ -e /etc/sysconfig/snmpd.options ] && echo 'OPTIONS="-LSwd -Lf /dev/null -p /var/run/snmpd.pid -a"'

# Launch SNMP server on startup:
chkconfig snmpd on
service snmpd restart
```

NRPE

Let's enable NRPE on our monitored hosts (port 5666).

```
# Activate the EPEL6 repository - install:
http://download.fedoraproject.org/pub/epel/6/i386/repoview/epel-release.html

# Install NRPE server:
yum install nrpe

# Allow access from Shinken poller:
sed -i -e 's/^allowed_hosts=.*/allowed_hosts=127.0.0.1,YOUR_SHINKEN_IP/' /etc/nagios/nrpe.cfg

# Launch NRPE server on startup:
chkconfig nrpe on
service nrpe start
```

Enable and configure remote checks in `/etc/nagios/nrpe.cfg`.

SSH

Let's give Shinken access to our monitored hosts, e.g. to execute event handlers or run NRPE through SSH:

On the Shinken Server, generate a SSH key `/home/shinken/.ssh/id_rsa`:

```
sudo -u shinken ssh-keygen</code>
```

On each monitored host:

- Create a ''monitaction" user with limited rights, accessed by Shinken:<code>

```
useradd -r monitaction -m
mkdir -pm 700 ~monitaction/.ssh/
echo "ssh-rsa AAAAB3...EKtMx/9o0ApJl shinken@rh6" > ~monitaction/.ssh/authorized_keys  # from /home/s
chown -R monitaction: ~monitaction/.ssh/
mkdir -pm 750 /etc/sudoers.d/
touch /etc/sudoers.d/local
chmod 440 /etc/sudoers.d/local
```

- Edit ''/etc/sudoers.d/local" to give it privileges, e.g.:

```
Defaults !requiretty
monitaction ALL= NOPASSWD: /sbin/service jbossas7 *
monitaction ALL= NOPASSWD: /sbin/service thunderhead *
monitaction ALL= NOPASSWD: /sbin/service httpd *
```

Test from the Shinken server:

```
ssh -oStrictHostKeyChecking=no -oUserKnownHostsFile=/dev/null 192.168.X.X -l monitaction -t sudo /sbi
```

Of course, open firewall access from the Shinken server to the monitored host's SSH.

Extra: Graphite

If you're interested in Graphite, you can start from this basis:

- *network_based_modules_-_graphite_graphing*
- *use_with_graphite*

Additional configuration:

```
echo "/opt/graphite/bin/carbon-cache.py start" >> /etc/rc.local
chgrp apache /opt/graphite/storage/
chmod g+w /opt/graphite/storage/
sudo -u apache /opt/graphite/bin/python /opt/graphite/webapp/graphite/manage.py runserver  # TODO: ad
# Remove the numerous dummy network graphs creating by mistake by Graphite:
echo "rm -f /opt/graphite/storage/whisper/*/shinken/NetworkUsage/*_13????????_.wsp" >> /etc/cron.dail
chmod 755 /etc/cron.daily/graphite-cleanup
```

2.3.8 GNU/Linux Installation from Source

Abstract

This guide is intended to provide you with simple instructions on how to install Shinken from source (code) on GNU/Linux and have it monitoring in no time. This the basic Shinken installation primarily meant for **packagers**.

Automated installation

Installation should be done using the *Shinken 10 minute installation*

- Installs Shinken
- Installs common checks scripts
- Installs core related optional modules
- Uses an **installation script** that will fetch all dependancies
- Internet access required

Find it at *Shinken 10 minute installation*.

Manual installation process for packagers

If you wish to install Shinken using the standalone setup.py installation method. Read on.

Operating system used for this walk through : Ubuntu Server edition 10.04 LTS

Requirements

Check the *Shinken Requirements*.

Make sure you've installed the required packages before continuing!

Create Shinken Account

Become the root user.

```
linux:~ $ sudo su -
```

Create a new shinken user account and give it a password.

```
linux:~ # /usr/sbin/useradd -m shinken
linux:~ # passwd shinken
```

On Ubuntu server edition (9.10 and newer versions), you will also need to add a shinken group (it's not created by default). You should be able to skip this step on desktop editions of Ubuntu.

```
linux:~ # /usr/sbin/groupadd shinken
linux:~ # /usr/sbin/usermod -G shinken shinken
```

Add the apache user to this group to allow external commands to be send from the web interface.

```
linux:~ # /usr/sbin/usermod -G shinken www-data
```

Download Shinken and the Plugins

Create a directory for storing the downloads.

```
linux:~ # mkdir ~/downloads
linux:~ # cd ~/downloads
```

Download the source code of Shinken and the Shinken plugins (visit http://www.nagios.org/download/ for links to the latest versions of the plugins). At the time of writing, the latest versions plugins were 1.4.13.

```
linux:~ # git clone git://shinken.git.sourceforge.net/gitroot/shinken/shinken
```

or

Get the latest official release: https://github.com/naparuba/shinken/tarball/1.0.1

Install Shinken

```
linux:~ # cd shinken
linux:~ # sudo python setup.py install --install-scripts=/usr/bin
```

Don't start Shinken yet - there's still more that needs to be done...

Customize the configuration

Sample *configuration files* have now been installed in the "/etc/shinken/" directory. These sample files should work fine for getting started with Shinken. You'll need to make just one change before you proceed.

Install the Nagios Plugins to use with Shinken

You can download plugins from source, but your debian-like administrator will just will you :

```
linux:~ # sudo apt-get install nagios-plugins
```

Start Shinken

Configure Shinken to automatically start when the system boots.

```
linux:~ # sudo ln -s /etc/init.d/shinken-scheduler /etc/rcS.d/S98shinken-scheduler
linux:~ # sudo ln -s /etc/init.d/shinken-poller /etc/rcS.d/S98shinken-poller
linux:~ # sudo ln -s /etc/init.d/shinken-reactionner /etc/rcS.d/S98shinken-reactionner
linux:~ # sudo ln -s /etc/init.d/shinken-broker /etc/rcS.d/S98shinken-broker
linux:~ # sudo ln -s /etc/init.d/shinken-arbiter /etc/rcS.d/S98shinken-arbiter
```

Verify the sample Shinken configuration files.

```
linux:~ # /usr/bin/shinken-arbiter -v -c /etc/shinken/nagios.cfg -c /etc/shinken/shinken-specific.cfg
```

If there are no errors, start Shinken.

```
linux:~ # sudo /etc/init.d/shinken-scheduler start
linux:~ # sudo /etc/init.d/shinken-poller start
linux:~ # sudo /etc/init.d/shinken-broker start
linux:~ # sudo /etc/init.d/shinken-reactionner start
linux:~ # sudo /etc/init.d/shinken-arbiter start
```

2.3.9 Windows Quickstart

Abstract

This guide is intended to provide you with simple instructions on how to install Shinken on windows and have it monitoring your local machine inside of 10 minutes. No advanced installation options are discussed here - just the basics that will work for 95% of users who want to get started.

These instructions were written based on a standard Windows (tested on 2k3, 2k8, XP and Seven).

Automated installation

Installation should be done using the *Shinken 10 minute installation*

- Installs Shinken
- Installs common checks scripts
- Installs core related optional modules
- Uses an **installation script** that will fetch all dependancies
- Internet access required

Read on to find out how to install Shinken manually.

Batch file manual installation process

Get the https://github.com/naparuba/shinken/zipball/1.2 file to c:shinken

During portions of the installation you will need to have administrator privileges to your server. Make sure you have installed the following packages on your Windows installation before continuing.

- Python 2.7 for Windows
- easy_install

  - add easy_install to your windows PATH as per setup tools instructions in link above
  - `easy_install pip`
- Pyro 4 library Windows
  - `pip pyro4=4.14`
- Windows server 2003 Resource Kit
- PyWin32

Take the files instsrv.exe and srvany.exe from the directory of the resource kit (typically "c:program filesWindows Resource KitsTools") and put them in the directory "c:shinkenwindows" (it should already exist by decompressing the archive, or you are a directory level to deep).

To install all services, launch the installation batch file:

```
c:\shinken\windows\install-all.bat
```

Launch services.msc to see your brand new services (Shinken-*).

You're Done

Congratulations! You sucessfully installed Shinken. Your journey into monitoring is just beginning. You'll no doubt want to monitor more than just your local machine, so check out the following docs...

- *Monitoring Windows machines*
- *Monitoring Linux/Unix machines*
- Monitoring Netware servers
- *Monitoring routers/switches*
- *Monitoring publicly available services ("HTTP", "FTP", "SSH", etc.)*

You can also follow the instructions on setting up the new monitoring template based method of monitoring your systems.

2.3.10 Nokia N900 Quickstart

This guide is intended to provide you with simple instructions on how to install Shinken from source (code) on a Nokia N900.

A Shinken how-to provides details on a new installation method using an install.sh script.

If you were really looking to install it on a GNU/Linux platform or Windows follow the *Shinken 10 minute installation guide.*.

If you really wish to install Shinken on a Nokia N900, Read On!

Required Packages

- Python >= 2.4
- Pyro (Python module for distributed objects)
- Git (If you want the latest code)

About Nokia N900

The smartphone Nokia N900 runs natively under the Maemo Operating System (which includes python 2.5.4). You can find more information about it at : http://maemo.org/intro/ and http://maemo.nokia.com/

Tip: You will need an internet connection to continu, don't forget to take care of your 3G data quota! It's easier to do it via ssh. And you will need of a tool like gainroot or rootsh to complete the following steps.

Create users :

```
Nokia-N900:~$ root
Nokia-N900:~#
Nokia-N900:~# useradd -m shinken
Nokia-N900:~# groupadd shinken
Nokia-N900:~# usermod -G shinken shinken
```

Then add theses two new repository at the end of /etc/apt/sources.list.d/hildon-application-manager.list :

```
deb http://repository.maemo.org/ fremantle/sdk free non-free
deb http://repository.maemo.org/ fremantle/tools free non-free
```

Next, install python an pyro :

```
Nokia-N900:~# aptitude update
Nokia-N900:~# aptitude install python-dev python-setuptools build-essential
Nokia-N900:~# easy_install http://www.xs4all.nl/~irmen/pyro3/download/Pyro-3.10.tar.gz
```

Now, we can install shinken, we use MyDocs because it's the mount point of the ssd drive :

```
Nokia-N900:~# cd /home/user/MyDocs/ && mkdir tmp && cd tmp
Nokia-N900:/home/user/MyDocs/tmp/# wget http://shinken-monitoring.org/pub/shinken-0.5.1.tar.gz
Nokia-N900:/home/user/MyDocs/tmp/# tar zxf shinken-0.5.1.tar.gz
Nokia-N900:/home/user/MyDocs/tmp/# cd shinken-0.5.1
Nokia-N900:/home/user/MyDocs/tmp/shinken-0.5.1# python setup.py install --install-scripts=/usr/bin/
Nokia-N900:/home/user/MyDocs/tmp/shinken-0.5.1# cp -af bin/* /usr/bin/
Nokia-N900:/home/user/MyDocs/tmp/shinken-0.5.1# sync && cd
```

You can test an see what happen :

```
Nokia-N900:~# launch_all.sh
Nokia-N900:~# ps aux | grep python
```

That's all! But for installation only! You need now to configure shinken.

Important: And don't forget to take care of your 3G data quota!!

2.4 Upgrading Shinken

2.4.1 Upgrading From Previous Shinken Releases

See the *update page for that*. Basically it's only about backuping and installing from a later git version.

2.4.2 Upgrading From Nagios 3.x

Just install Shinken and start the arbiter with your Nagios configuration. That's all.

2.5 Monitoring Windows Machines

Abstract

This document describes how you can monitor "private" services and attributes of Windows machines, such as:

- Memory usage
- CPU load
- Disk usage
- Service states
- Running processes
- etc.

2.5.1 Introduction

Publicly available services that are provided by Windows machines ("HTTP", "FTP", "POP3", etc.) can be monitored easily by following the documentation on *Monitoring publicly available services (HTTP, FTP, SSH, etc.)*.

These instructions assume that you've installed Shinken according to the *quickstart guide*. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you follow the quickstart.

2.5.2 Overview

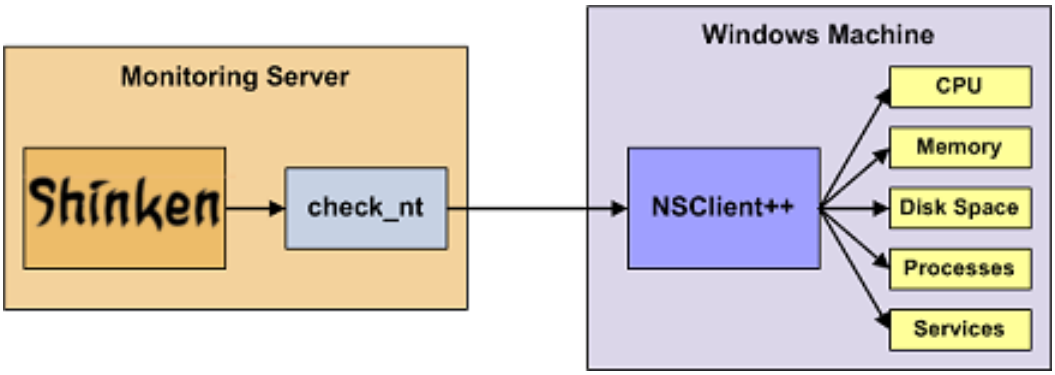

Monitoring private services or attributes of a Windows machine requires that you install an agent on it. This agent acts as a proxy between the Nagios plugin that does the monitoring and the actual service or attribute of the Windows machine. Without installing an agent on the Windows box, Shinken would be unable to monitor private services or attributes of the Windows box.

For this example, we will be installing the NSClient++ addon on the Windows machine and using the **check_nt** plugin to communicate with the NSClient++ addon. The **check_nt** plugin should already be installed on the Shinken server if you followed the quickstart guide.

Other Windows agents (like NC_Net) could be used instead of NSClient++ if you wish - provided you change command and service definitions, etc. a bit. For the sake of simplicity I will only cover using the NSClient++ addon in these instructions.

2.5.3 Steps

There are several steps you'll need to follow in order to monitor a new Windows machine. They are:

1. Perform first-time prerequisites
2. Install a monitoring agent on the Windows machine
3. Create new host and service definitions for monitoring the Windows machine
4. Restart the Shinken daemon

2.5.4 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A **check_nt** command definition has been added to the "commands.cfg" file. This allows you to use the **check_nt** plugin to monitor Window services
- Windows server host template (called "windows-server") has already been created in the "templates.cfg" file. This allows you to add new Windows host definitions in a simple manner

The above-mentioned config files can be found in the "/usr/local/nagios/etc/objects/" directory. You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Nagios before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your Windows boxes in no time.

2.5.5 Prerequisites

The first time you configure Shinken to monitor a Windows machine, you'll need to do a bit of extra work. Remember, you only need to do this for the *first* Windows machine you monitor.

Edit the main Shinken config file.

```
linux:~ # vi /usr/local/shinken/etc/nagios.cfg
```

Remove the leading pound (#) sign from the following line in the main configuration file:

```
#cfg_file=/usr/local/shinken/etc/objects/windows.cfg
```

Save the file and exit.

What did you just do? You told Shinken to look to the "/usr/local/shinken/etc/objects/windows.cfg" to find additional object definitions. That's where you'll be adding Windows host and service definitions. That configuration file already contains some sample host, hostgroup, and service definitions. For the *first* Windows machine you monitor, you can simply modify the sample host and service definitions in that file, rather than creating new ones.

2.5.6 Installing the Windows Agent

Before you can begin monitoring private services and attributes of Windows machines, you'll need to install an agent on those machines. I recommend using the NSClient++ addon, which can be found at http://sourceforge.net/projects/nscplus. These instructions will take you through a basic installation of the NSClient++ addon, as well as the configuration of Shinken for monitoring the Windows machine.

1. Download the latest stable version of the NSClient++ addon from http://sourceforge.net/projects/nscplus
2. Unzip the NSClient++ files into a new C:NSClient++ directory
3. Open a command prompt and change to the C:NSClient++ directory

4. Register the NSClient++ system service with the following command:

```
C:\> nsclient++ /install
```

5. Install the NSClient++ systray with the following command ('SysTray' is case-sensitive):

```
C:\> nsclient++ SysTray
```

6. Open the services manager and make sure the NSClientpp service is allowed to interact with the desktop (see the 'Log On' tab of the services manager). If it isn't already allowed to interact with the desktop, check the box to allow it to.

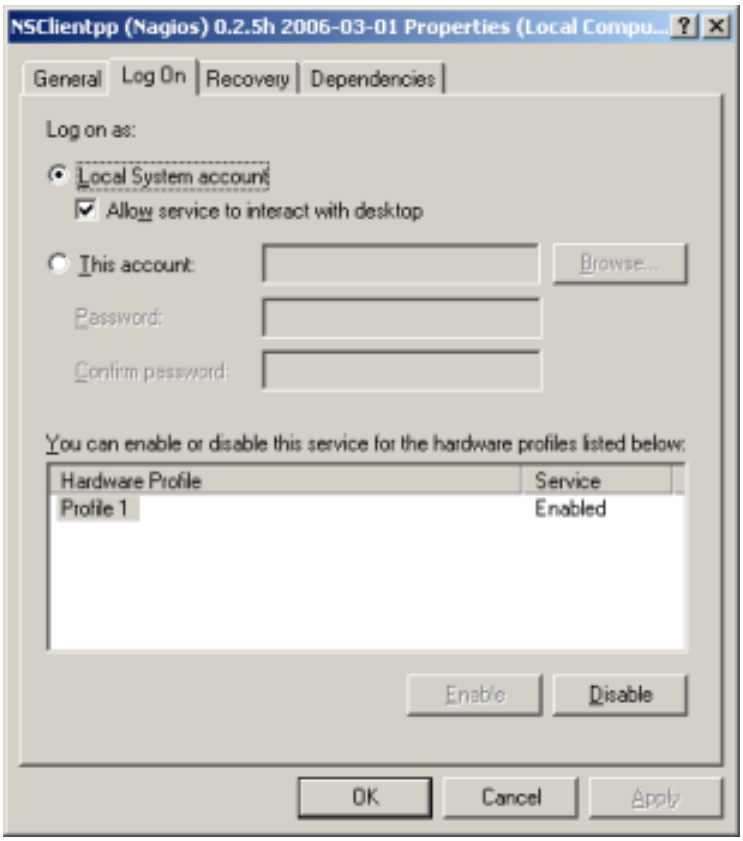

7. Edit the "NSC.INI file" (located in the "C:NSClient++" directory) and make the following changes:

- Uncomment all the modules listed in the [modules] section, except for "CheckWMI.dll" and "RemoteConfiguration.dll"
- Optionally require a password for clients by changing the "password" option in the [Settings] section.
- Uncomment the "allowed_hosts" option in the [Settings] section. Add the IP address of the Nagios server to this line, or leave it blank to allow all hosts to connect.
- Make sure the "port" option in the [NSClient] section is uncommented and set to '12489' (the default port).

8. Start the NSClient++ service with the following command:

```
C:\> nsclient++ /start
```

9. If installed properly, a new icon should appear in your system tray. It will be a yellow circle with a black 'M' inside.

10. Success! The Windows server can now be added to the Shinken monitoring configuration...

2.5.7 Configuring Shinken

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Windows machine.

Open the "windows.cfg" file for editing.

```
linux:~ # vi /etc/shinken/objects/windows.cfg
```

Add a new *host* definition for the Windows machine that you're going to monitor. If this is the *first* Windows machine you're monitoring, you can simply modify the sample host definition in "windows.cfg". Change the "host_name", "alias", and "address" fields to appropriate values for the Windows box.

```
define host{
    use     windows-server  ; Inherit default values from a Windows server template (make sure you ke
    host_name       winserver
    alias       My Windows Server
    address     192.168.1.2
}
```

Good. Now you can add some service definitions (to the same configuration file) in order to tell Shinken to monitor different aspects of the Windows machine. If this is the *first* Windows machine you're monitoring, you can simply modify the sample service definitions in "windows.cfg".

Replace ""*"winserver"*"" in the example definitions below with the name you specified in the "host_name" directive of the host definition you just added.

Add the following service definition to monitor the version of the NSClient++ addon that is running on the Windows server. This is useful when it comes time to upgrade your Windows servers to a newer version of the addon, as you'll be able to tell which Windows machines still need to be upgraded to the latest version of NSClient++.

```
define service{
    use         generic-service
    host_name           winserver
    service_description NSClient++ Version
    check_command       check_nt!CLIENTVERSION
}
```

Add the following service definition to monitor the uptime of the Windows server.

```
define service{
    use         generic-service
    host_name           winserver
    service_description Uptime
    check_command       check_nt!UPTIME
}
```

Add the following service definition to monitor the CPU utilization on the Windows server and generate a CRITICAL alert if the 5-minute CPU load is 90% or more or a WARNING alert if the 5-minute load is 80% or greater.

```
define service{
    use         generic-service
    host_name           winserver
    service_description CPU Load
    check_command       check_nt!CPULOAD!-l 5,80,90
}
```

Add the following service definition to monitor memory usage on the Windows server and generate a CRITICAL alert if memory usage is 90% or more or a WARNING alert if memory usage is 80% or greater.

```
define service{
    use         generic-service
    host_name           winserver
    service_description Memory Usage
    check_command       check_nt!MEMUSE!-w 80 -c 90
}
```

Add the following service definition to monitor usage of the C:drive on the Windows server and generate a CRITICAL alert if disk usage is 90% or more or a WARNING alert if disk usage is 80% or greater.

```
define service{
    use         generic-service
    host_name           winserver
    service_description C:\ Drive Space
    check_command       check_nt!USEDDISKSPACE!-l c -w 80 -c 90
}
```

Add the following service definition to monitor the W3SVC service state on the Windows machine and generate a CRITICAL alert if the service is stopped.

```
define service{
    use         generic-service
    host_name           winserver
    service_description W3SVC
    check_command       check_nt!SERVICESTATE!-d SHOWALL -l W3SVC
}
```

Add the following service definition to monitor the Explorer.exe process on the Windows machine and generate a CRITICAL alert if the process is not running.

```
define service{
    use         generic-service
    host_name           winserver
    service_description Explorer
    check_command       check_nt!PROCSTATE!-d SHOWALL -l Explorer.exe
}
```

That's it for now. You've added some basic services that should be monitored on the Windows box. Save the configuration file.

2.5.8 Password Protection

If you specified a password in the NSClient++ configuration file on the Windows machine, you'll need to modify the **check_nt** command definition to include the password. Open the "commands.cfg" file for editing.

```
linux:~ # vi /usr/local/nagios/etc/commands.cfg
```

Change the definition of the **check_nt** command to include the ""-s" <PASSWORD>" argument (where PASSWORD is the password you specified on the Windows machine) like this:

```
define command {
    command_name    check_nt
    command_line    $USER1$/check_nt -H $HOSTADDRESS$ -p 12489 -s PASSWORD -v $ARG1$ $ARG2$
}
```

Save the file.

2.5.9 Restarting Shinken

You're done with modifying the Shiknen configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

2.6 Monitoring GNU/Linux & Unix Machines

Abstract

This document describes how you can monitor "private" services and attributes of Linux/UNIX servers, such as:

- CPU load
- Memory usage
- Disk usage
- Logged in users
- Running processes
- etc.

2.6.1 Introduction

Publicly available services that are provided by Linux servers ("HTTP", "FTP", "SSH", "SMTP", etc.) can be monitored easily by following the documentation on *Monitoring publicly available services*.

These instructions assume that you've installed Shinken according to the *quickstart guide*. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you follow the quickstart.

2.6.2 Overview

[This document has not been completed. I would recommend you read the documentation on the *NRPE addon* for instructions on how to monitor a remote Linux/Unix server.]

There are several different ways to monitor attributes or remote Linux/Unix servers. One is by using shared "SSH" keys and the **check_by_ssh** plugin to execute plugins on remote servers. This method will not be covered here, but can result in high load on your monitoring server if you are monitoring hundreds or thousands of services. The overhead of setting up/destroying "SSH" connections is the cause of this.

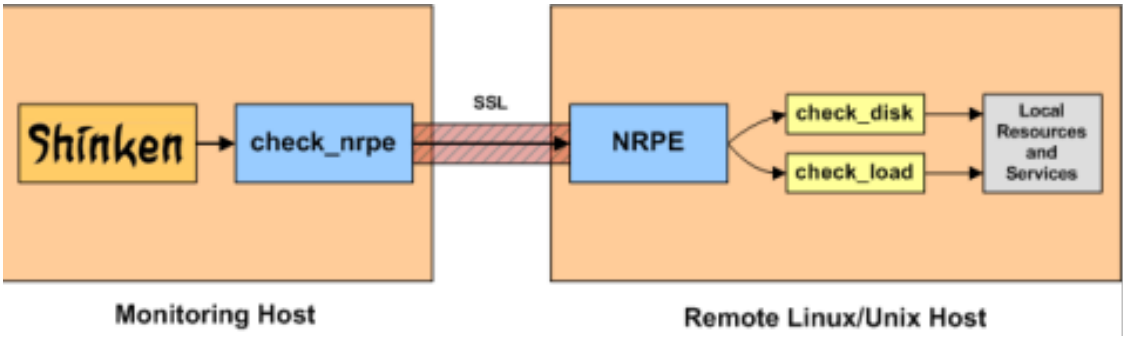

Another common method of monitoring remote Linux/Unix hosts is to use the *NRPE addon*. NRPE allows you to execute plugins on remote Linux/Unix hosts. This is useful if you need to monitor local resources/attributes like disk usage, CPU load, memory usage, etc. on a remote host.

2.7 Monitoring Network Printers

Abstract

This document describes how you can monitor the status of networked printers. Specifically, HP™ printers that have internal/external JetDirect® cards/devices, or other print servers (like the Troy™ PocketPro 100S® or the Netgear™ PS101®) that support the JetDirect protocol.

2.7.1 Introduction

The **check_hpjd** plugin (which is part of the standard Nagios plugins distribution) allows you to monitor the status of JetDirect-capable printers which have "SNMP" enabled. The plugin is capable of detecting the following printer states:

- Paper Jam
- Out of Paper
- Printer Offline
- Intervention Required
- Toner Low
- Insufficient Memory
- Open Door
- Output Tray is Full
- and more...

These instructions assume that you've installed Nagios according to the *quickstart guide*. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you follow the quickstart.

2.7.2 Overview

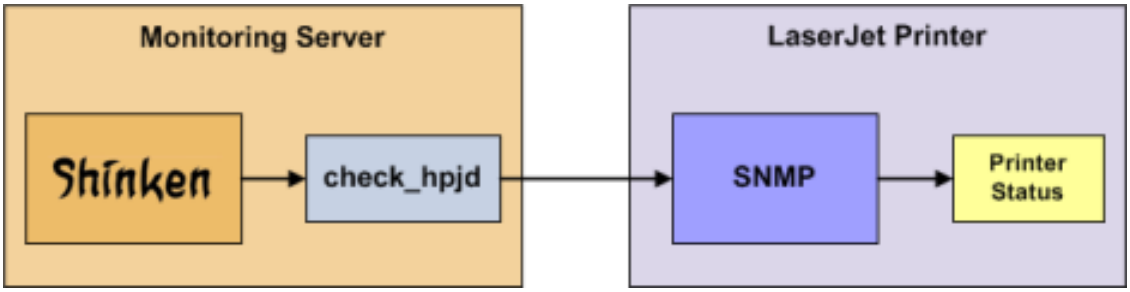

Monitoring the status of a networked printer is pretty simple. JetDirect-enabled printers usually have "SNMP" enabled, which allows Nagios to monitor their status using the **check_hpjd** plugin.

The **check_hpjd** plugin will only get compiled and installed if you have the net-snmp and net-snmp-utils packages installed on your system. Make sure the plugin exists in "/usr/local/nagios/libexec" before you continue. If it doesn't, install net-snmp and net-snmp-utils and recompile/reinstall the Nagios plugins.

2.7.3 Steps

There are several steps you'll need to follow in order to monitor a new network printer. They are:

- Perform first-time prerequisites
- Create new host and service definitions for monitoring the printer
- Restart Shinken services

2.7.4 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A *check_hpjd* command definition has been added to the "commands.cfg" file. This allows you to use the **check_hpjd** plugin to monitor network printers.
- A printer host template (called *generic-printer*) has already been created in the "templates.cfg" file. This allows you to add new printer host definitions in a simple manner.

The above-mentioned config files can be found in the "/etc/shinken/objects/" directory. You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your network printers in no time.

2.7.5 Prerequisites

The first time you configure Shinken to monitor a network printer, you'll need to do a bit of extra work. Remember, you only need to do this for the *first* printer you monitor.

Edit the main Nagios config file.

```
linux:~ # vi /etc/shinken/nagios.cfg
```

Remove the leading pound (#) sign from the following line in the main configuration file:

```
#cfg_file=/etc/shinken/objects/printer.cfg
```

Save the file and exit.

What did you just do? You told Shinken to look to the "/etc/shinken/objects/printer.cfg" to find additional object definitions. That's where you'll be adding host and service definitions for the printer. That configuration file already contains some sample host, hostgroup, and service definitions. For the *first* printer you monitor, you can simply modify the sample host and service definitions in that file, rather than creating new ones.

2.7.6 Configuring Shinken

You'll need to create some *object definitions* in order to monitor a new printer.

Open the "printer.cfg" file for editing.

```
linux:~ # vi /etc/shinken/objects/printer.cfg
```

Add a new *host* definition for the networked printer that you're going to monitor. If this is the *first* printer you're monitoring, you can simply modify the sample host definition in "printer.cfg". Change the "host_name", "alias", and "address" fields to appropriate values for the printer.

```
define host{
    use         generic-printer     ; Inherit default values from a template
    host_name   hplj2605dn          ; The name we're giving to this printer
    alias       HP LaserJet 2605dn  ; A longer name associated with the printer
    address     192.168.1.30        ; IP address of the printer
    hostgroups  allhosts            ; Host groups this printer is associated with
}
```

Now you can add some service definitions (to the same configuration file) to monitor different aspects of the printer. If this is the *first* printer you're monitoring, you can simply modify the sample service definition in "printer.cfg".

Replace "hplj2605dn" in the example definitions below with the name you specified in the "host_name" directive of the host definition you just added.

Add the following service definition to check the status of the printer. The service uses the **check_hpjd** plugin to check the status of the printer every 10 minutes by default. The "SNMP" community string used to query the printer is "public" in this example.

```
define service{
    use                   generic-service        ; Inherit values from a template
    host_name             hplj2605dn             ; The name of the host the service is associated wit
    service_description   Printer Status         ; The service description
    check_command         check_hpjd!-C public   ; The command used to monitor the service
    normal_check_interval 10  ; Check the service every 10 minutes under normal conditions
    retry_check_interval  1   ; Re-check the service every minute until its final/hard state is deter
}
```

Add the following service definition to ping the printer every 10 minutes by default. This is useful for monitoring RTA, packet loss, and general network connectivity.

```
define service{
        use                     generic-service
        host_name               hplj2605dn
        service_description     PING
        check_command           check_ping!3000.0,80%!5000.0,100%
        normal_check_interval   10
        retry_check_interval    1
}
```

Save the file.

2.7.7 Restarting Shinken

Once you've added the new host and service definitions to the "printer.cfg" file, you're ready to start monitoring the printer. To do this, you'll need to *verify your configuration* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

2.8 Monitoring Routers and Switches

Abstract

This document describes how you can monitor the status of network switches and routers. Some cheaper "unmanaged" switches and hubs don't have IP addresses and are essentially invisible on your network, so there's not any way to monitor them. More expensive switches and routers have addresses assigned to them and can be monitored by pinging them or using "SNMP" to query status information.

2.8.1 Introduction

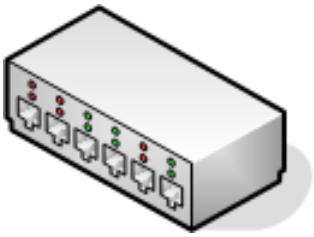

I'll describe how you can monitor the following things on managed switches, hubs, and routers:

- Packet loss, round trip average
- "SNMP" status information
- Bandwidth / traffic rate

These instructions assume that you've installed Shinken according to the *quickstart guide*. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed when you follow the quickstart.

2.8.2 Overview

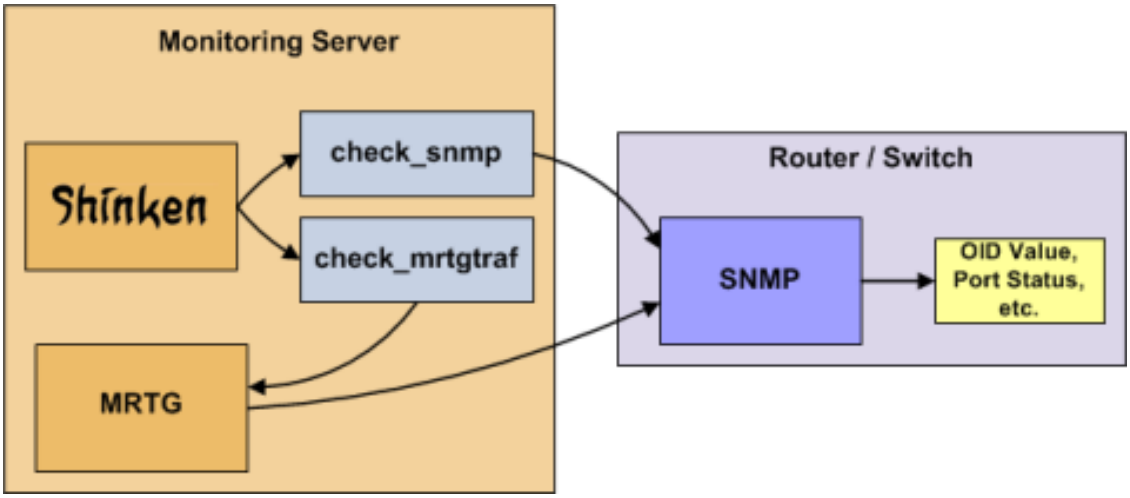

Monitoring switches and routers can either be easy or more involved - depending on what equipment you have and what you want to monitor. As they are critical infrastructure components, you'll no doubt want to monitor them in at least some basic manner.

Switches and routers can be monitored easily by "pinging" them to determine packet loss, RTA, etc. If your switch supports "SNMP", you can monitor port status, etc. with the **check_snmp** plugin and bandwidth (if you're using MRTG) with the **check_mrtgtraf** plugin.

The **check_snmp** plugin will only get compiled and installed if you have the net-snmp and net-snmp-utils packages installed on your system. Make sure the plugin exists in "/usr/local/nagios/libexec" before you continue. If it doesn't, install net-snmp and net-snmp-utils and recompile/reinstall the Nagios plugins.

2.8.3 Steps

There are several steps you'll need to follow in order to monitor a new router or switch. They are:

- Perform first-time prerequisites
- Create new host and service definitions for monitoring the device
- Restart Shinken services

2.8.4 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- Two command definitions (*check_snmp* and *check_local_mrtgtraf*) have been added to the "commands.cfg" file. These allows you to use the **check_snmp** and **check_mrtgtraf** plugins to monitor network routers.
- A switch host template (called *generic-switch*) has already been created in the "templates.cfg" file. This allows you to add new router/switch host definitions in a simple manner.

The above-mentioned config files can be found in the "/etc/shinken/objects/" directory. You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your network routers/switches in no time.

Important: The commands are in fact not included yet in commands.cfg

2.8.5 Prerequisites

The first time you configure Shinken to monitor a network switch, you'll need to do a bit of extra work. Remember, you only need to do this for the *first* switch you monitor.

Edit the main Shinken config file.

```
linux:~ # vi /etc/shinken/nagios.cfg
```

Remove the leading pound (#) sign from the following line in the main configuration file:

```
#cfg_file=/etc/shinken/objects/switch.cfg
```

Save the file and exit.

What did you just do? You told Shinken to look to the "/etc/shinken/objects/switch.cfg" to find additional object definitions. That's where you'll be adding host and service definitions for routers and switches. That configuration file already contains some sample host, hostgroup, and service definitions. For the *first* router/switch you monitor, you can simply modify the sample host and service definitions in that file, rather than creating new ones.

2.8.6 Configuring Shinken

You'll need to create some *object definitions* in order to monitor a new router/switch.

Open the "switch.cfg" file for editing.

```
linux:~ # vi /etc/shinken/objects/switch.cfg
```

Add a new *host* definition for the switch that you're going to monitor. If this is the *first* switch you're monitoring, you can simply modify the sample host definition in "switch.cfg". Change the "host_name", "alias", and "address" fields to appropriate values for the switch.

```
define host{
    use        generic-switch         ; Inherit default values from a template
    host_name  linksys-srw224p        ; The name we're giving to this switch
    alias      Linksys SRW224P Switch ; A longer name associated with the switch
    address    192.168.1.253          ; IP address of the switch
    hostgroups allhosts,switches      ; Host groups this switch is associated with
}
```

2.8.7 Monitoring Services

Now you can add some service definitions (to the same configuration file) to monitor different aspects of the switch. If this is the *first* switch you're monitoring, you can simply modify the sample service definition in "switch.cfg".

Replace *linksys-srw224p* in the example definitions below with the name you specified in the "host_name" directive of the host definition you just added.

2.8.8 Monitoring Packet Loss and RTA

Add the following service definition in order to monitor packet loss and round trip average between the Shinken host and the switch every 5 minutes under normal conditions.

```
define service{
   use                    generic-service
   host_name              linksys-srw224p
   service_description    PING
   check_command          check_ping!200.0,20%!600.0,60%
   normal_check_interval  5
   retry_check_interval   1
}
```

1. Inherit values from a template
2. The name of the host the service is associated with
3. The service description
4. The command used to monitor the service
5. Check the service every 5 minutes under normal conditions
6. Re-check the service every minute until its final/hard state is determined

This service will be:

- CRITICAL if the round trip average (RTA) is greater than 600 milliseconds or the packet loss is 60% or more
- WARNING if the RTA is greater than 200 ms or the packet loss is 20% or more
- OK if the RTA is less than 200 ms and the packet loss is less than 20%

2.8.9 Monitoring SNMP Status Information

If your switch or router supports "SNMP", you can monitor a lot of information by using the **check_snmp** plugin. If it doesn't, skip this section.

Add the following service definition to monitor the uptime of the switch.

```
define service{
    use                  generic-service ; Inherit values from a template
    host_name            linksys-srw224p
    service_description  Uptime
    check_command        check_snmp!-C public -o sysUpTime.0
}
```

In the "check_command" directive of the service definition above, the "-C public" tells the plugin that the "SNMP" community name to be used is "public" and the "-o sysUpTime.0" indicates which OID should be checked.

If you want to ensure that a specific port/interface on the switch is in an up state, you could add a service definition like this:

```
define service{
    use                 generic-service ; Inherit values from a template
    host_name           linksys-srw224p
    service_description Port 1 Link Status
    check_command       check_snmp!-C public -o ifOperStatus.1 -r 1 -m RFC1213-MIB
}
```

In the example above, the "-o ifOperStatus.1" refers to the OID for the operational status of port 1 on the switch.

The "-r 1" option tells the **check_snmp** plugin to return an OK state if "1" is found in the "SNMP" result (1 indicates an "up" state on the port) and CRITICAL if it isn't found.

The "-m RFC1213-MIB" is optional and tells the **check_snmp** plugin to only load the "RFC1213-MIB" instead of every single MIB that's installed on your system, which can help speed things up.

That's it for the "SNMP" monitoring example. There are a million things that can be monitored via "SNMP", so its up to you to decide what you need and want to monitor. Good luck!

You can usually find the OIDs that can be monitored on a switch by running the following command (replace *192.168.1.253* with the IP address of the switch): **snmpwalk -v1 -c public 192.168.1.253 -m ALL .1**

2.8.10 Monitoring Bandwidth / Traffic Rate

If you're monitoring bandwidth usage on your switches or routers using MRTG, you can have Shinken alert you when traffic rates exceed thresholds you specify. The **check_mrtgtraf** plugin (which is included in the Nagios plugins distribution) allows you to do this.

You'll need to let the **check_mrtgtraf** plugin know what log file the MRTG data is being stored in, along with thresholds, etc. In my example, I'm monitoring one of the ports on a Linksys switch. The MRTG log file is stored in "/var/lib/mrtg/192.168.1.253_1.log". Here's the service definition I use to monitor the bandwidth data that's stored in the log file...

```
define service{
   use                 generic-service ; Inherit values from a template
   host_name           linksys-srw224p
   service_description Port 1 Bandwidth Usage
   check_command       check_local_mrtgtraf!/var/lib/mrtg/192.168.1.253_1.log!AVG!1000000,2000000!500
}
```

In the example above, the "/var/lib/mrtg/192.168.1.253_1.log" option that gets passed to the *check_local_mrtgtraf* command tells the plugin which MRTG log file to read from.

The AVG option tells it that it should use average bandwidth statistics. The "1000000,2000000" options are the warning thresholds (in bytes) for incoming traffic rates.

The "5000000,5000000" are critical thresholds (in bytes) for outgoing traffic rates. The "10" option causes the plugin to return a CRITICAL state if the MRTG log file is older than 10 minutes (it should be updated every 5 minutes).

Save the file.

2.8.11 Restarting Shinken

Once you've added the new host and service definitions to the "switch.cfg" file, you're ready to start monitoring the router/switch. To do this, you'll need to *verify your configuration* and *restart Sinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

2.9 Monitoring Publicly Available Services

Abstract

This document describes how you can monitor publicly available services, applications and protocols. By "public" I mean services that are accessible across the network - either the local network or the greater Internet. Examples of

public services include "HTTP", "POP3", "IMAP", "FTP", and "SSH". There are many more public services that you probably use on a daily basis. These services and applications, as well as their underlying protocols, can usually be monitored by Shinken without any special access requirements.

2.9.1 Introduction

Private services, in contrast, cannot be monitored with Shinken without an intermediary agent of some kind. Examples of private services associated with hosts are things like CPU load, memory usage, disk usage, current user count, process information, etc. These private services or attributes of hosts are not usually exposed to external clients. This situation requires that an intermediary monitoring agent be installed on any host that you need to monitor such information on. More information on monitoring private services on different types of hosts can be found in the documentation on:

- *Monitoring Windows machines*
- Monitoring Netware servers
- *Monitoring Linux/Unix machines*

Occasionally you will find that information on private services and applications can be monitored with "SNMP". The "SNMP" agent allows you to remotely monitor otherwise private (and inaccessible) information about the host. For more information about monitoring services using "SNMP", check out the documentation on *Monitoring routers/switches*.

These instructions assume that you've installed Shinken according to the *quickstart guide*. The sample configuration entries below reference objects that are defined in the sample "commands.cfg" and "localhost.cfg" config files.

2.9.2 Plugins For Monitoring Services

When you find yourself needing to monitor a particular application, service, or protocol, chances are good that a *plugin* exists to monitor it. The official Nagios plugins distribution comes with plugins that can be used to monitor a variety of services and protocols. There are also a large number of contributed plugins that can be found in the "contrib/" subdirectory of the plugin distribution. The NagiosExchange.org website hosts a number of additional plugins that have been written by users, so check it out when you have a chance.

If you don't happen to find an appropriate plugin for monitoring what you need, you can always write your own. Plugins are easy to write, so don't let this thought scare you off. Read the documentation on *developing plugins* for more information.

I'll walk you through monitoring some basic services that you'll probably use sooner or later. Each of these services can be monitored using one of the plugins that gets installed as part of the Nagios plugins distribution. Let's get started...

2.9.3 Creating A Host Definition

Before you can monitor a service, you first need to define a *host* that is associated with the service. You can place host definitions in any object configuration file specified by a *cfg_file* directive or placed in a directory specified by a *cfg_dir* directive. If you have already created a host definition, you can skip this step.

For this example, lets say you want to monitor a variety of services on a remote host. Let's call that host *remotehost*. The host definition can be placed in its own file or added to an already exiting object configuration file. Here's what the host definition for *remotehost* might look like:

```
define host{
    use             generic-host        ; Inherit default values from a template
    host_name       remotehost          ; The name we're giving to this host
    alias           Some Remote Host    ; A longer name associated with the host
    address         192.168.1.50        ; IP address of the host
    hostgroups      allhosts            ; Host groups this host is associated with
}
```

Now that a definition has been added for the host that will be monitored, we can start defining services that should be monitored. As with host definitions, service definitions can be placed in any object configuration file.

2.9.4 Creating Service Definitions

For each service you want to monitor, you need to define a *service* in Shinken that is associated with the host definition you just created. You can place service definitions in any object configuration file specified by a *cfg_file* directive or placed in a directory specified by a *cfg_dir* directive.

Some example service definitions for monitoring common public service ("HTTP", "FTP", etc) are given below.

2.9.5 Monitoring HTTP

Chances are you're going to want to monitor web servers at some point - either yours or someone else's. The **check_http** plugin is designed to do just that. It understands the "HTTP" protocol and can monitor response time, error codes, strings in the returned HTML, server certificates, and much more.

The "commands.cfg" file contains a command definition for using the **check_http** plugin. It looks like this:

```
define command{
    name            check_http
    command_name    check_http
    command_line    $USER1$/check_http -I $HOSTADDRESS$ $ARG1$
    }
```

A simple service definition for monitoring the "HTTP" service on the *remotehost* machine might look like this:

```
define service{
    use                 generic-service     ; Inherit default values from a template
    host_name           remotehost
    service_description HTTP
    check_command       check_http
    }
```

This simple service definition will monitor the "HTTP" service running on *remotehost*. It will produce alerts if the web server doesn't respond within 10 seconds or if it returns "HTTP" errors codes (403, 404, etc.). That's all you need for basic monitoring. Pretty simple, huh?

For more advanced monitoring, run the **check_http** plugin manually with "–help" as a command-line argument to see all the options you can give the plugin. This "–help" syntax works with all of the plugins I'll cover in this document.

A more advanced definition for monitoring the "HTTP" service is shown below. This service definition will check to see if the /download/index.php URI contains the string "latest-version.tar.gz". It will produce an error if the string isn't found, the URI isn't valid, or the web server takes longer than 5 seconds to respond.

```
define service{
    use                 generic-service   ; Inherit default values from a template
    host_name           remotehost
    service_description Product Download Link
```

```
    check_command       check_http!-u /download/index.php -t 5 -s "latest-version.tar.gz"
    }
```

2.9.6 Monitoring FTP

When you need to monitor "FTP" servers, you can use the **check_ftp** plugin. The "commands.cfg" file contains a command definition for using the **check_ftp** plugin, which looks like this:

```
define command{
    command_name    check_ftp
    command_line    $USER1$/check_ftp -H $HOSTADDRESS$ $ARG1$
    }
```

A simple service definition for monitoring the "FTP" server on *remotehost* would look like this:

```
define service{
    use                   generic-service  ; Inherit default values from a template
    host_name             remotehost
    service_description   FTP
    check_command         check_ftp
    }
```

This service definition will monitor the "FTP" service and generate alerts if the "FTP" server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the "FTP" server running on port 1023 on *remotehost*. It will generate an alert if the server doesn't respond within 5 seconds or if the server response doesn't contain the string "Pure-FTPd [TLS]".

```
define service{
    use                   generic-service   ; Inherit default values from a template
    host_name             remotehost
    service_description   Special FTP
    check_command         check_ftp!-p 1023 -t 5 -e "Pure-FTPd [TLS]"
    }
```

2.9.7 Monitoring SSH

When you need to monitor "SSH" servers, you can use the **check_ssh** plugin. The "commands.cfg" file contains a command definition for using the **check_ssh** plugin, which looks like this:

```
define command{
    command_name    check_ssh
    command_line    $USER1$/check_ssh $ARG1$ $HOSTADDRESS$
    }
```

A simple service definition for monitoring the "SSH" server on *remotehost* would look like this:

```
define service{
    use                  generic-service  ; Inherit default values from a template
    host_name            remotehost
    service_description  SSH
    check_command        check_ssh
    }
```

This service definition will monitor the "SSH" service and generate alerts if the "SSH" server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the "SSH" server and generate an alert if the server doesn't respond within 5 seconds or if the server version string string doesn't match "OpenSSH_4.2".

```
define service{
    use                 generic-service   ; Inherit default values from a template
    host_name           remotehost
    service_description SSH Version Check
    check_command       check_ssh!-t 5 -r "OpenSSH_4.2"
    }
```

2.9.8 Monitoring SMTP

The **check_smtp** plugin can be using for monitoring your email servers. The "commands.cfg" file contains a command definition for using the **check_smtp** plugin, which looks like this:

```
define command{
    command_name    check_smtp
    command_line    $USER1$/check_smtp -H $HOSTADDRESS$ $ARG1$
    }
```

A simple service definition for monitoring the "SMTP" server on *remotehost* would look like this:

```
define service{
    use                  generic-service  ; Inherit default values from a template
    host_name            remotehost
    service_description  SMTP
    check_command        check_smtp
    }
```

This service definition will monitor the "SMTP" service and generate alerts if the "SMTP" server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the "SMTP" server and generate an alert if the server doesn't respond within 5 seconds or if the response from the server doesn't contain "mygreatmailserver.com".

```
define service{
    use                  generic-service  ; Inherit default values from a template
    host_name            remotehost
    service_description  SMTP Response Check
    check_command        check_smtp!-t 5 -e "mygreatmailserver.com"
    }
```

2.9.9 Monitoring POP3

The **check_pop** plugin can be using for monitoring the "POP3" service on your email servers. The "commands.cfg" file contains a command definition for using the **check_pop** plugin, which looks like this:

```
define command{
    command_name    check_pop
    command_line    $USER1$/check_pop -H $HOSTADDRESS$ $ARG1$
    }
```

A simple service definition for monitoring the "POP3" service on *remotehost* would look like this:

```
define service{
    use                     generic-service  ; Inherit default values from a template
    host_name               remotehost
    service_description     POP3
    check_command           check_pop
    }
```

This service definition will monitor the "POP3" service and generate alerts if the "POP3" server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the "POP3" service and generate an alert if the server doesn't respond within 5 seconds or if the response from the server doesn't contain "mygreatmailserver.com".

```
define service{
    use                     generic-service  ; Inherit default values from a template
    host_name               remotehost
    service_description     POP3 Response Check
    check_command           check_pop!-t 5 -e "mygreatmailserver.com"
    }
```

2.9.10 Monitoring IMAP

The **check_imap** plugin can be using for monitoring "IMAP4" service on your email servers. The "commands.cfg" file contains a command definition for using the **check_imap** plugin, which looks like this:

```
define command{
    command_name    check_imap
    command_line    $USER1$/check_imap -H $HOSTADDRESS$ $ARG1$
    }
```

A simple service definition for monitoring the "IMAP4" service on *remotehost* would look like this:

```
define service{
    use                     generic-service  ; Inherit default values from a template
    host_name               remotehost
    service_description     IMAP
    check_command           check_imap
    }
```

This service definition will monitor the "IMAP4" service and generate alerts if the "IMAP" server doesn't respond within 10 seconds.

A more advanced service definition is shown below. This service will check the IAMP4 service and generate an alert if the server doesn't respond within 5 seconds or if the response from the server doesn't contain "mygreatmailserver.com".

```
define service{
    use                     generic-service  ; Inherit default values from a template
    host_name               remotehost
    service_description     IMAP4 Response Check
    check_command           check_imap!-t 5 -e "mygreatmailserver.com"
    }
```

2.9.11 Restarting Shinken

Once you've added the new host and service definitions to your object configuration file(s), you're ready to start monitoring them. To do this, you'll need to *verify your configuration* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

2.10 Nagios/Shinken Plugins

2.11 Introduction

Shinken includes a set of scalable internal mechanisms for checking the status of hosts and services on your network. These are called modules and can be loaded by the various Shinken daemons involved in data acquisition (Poller daemons, Receiver daemons, Arbiter Daemon) Shinken also relies on external programs (called check plugins) to monitor a very wide variety of devices, applications and networked services.

2.12 What Are Plugins?

Plugins are compiled executables or scripts (Perl scripts, shell scripts, etc.) that can be run from a command line to check the status of a host or service. Shinken uses the results from plugins to determine the current status of hosts and services on your network and obtain performance data about the monitored service.

Shinken will execute a plugin whenever there is a need to check the status of a service or host. The plugin does something (notice the very general term) to perform the check and then simply returns the results to Shinken. It will process the results that it receives from the plugin and take any necessary actions (running *event handlers*, sending out *notifications*, etc).

2.13 Shinken integrated data acquisition modules

These replace traditional unscalable plugins with high performance variants that are more tightly coupled with Shinken.

Integrated Shinken data acquisition modules support the following protocols:

- *NRPE*
- *SNMP*

2.14 Plugins As An Abstraction Layer

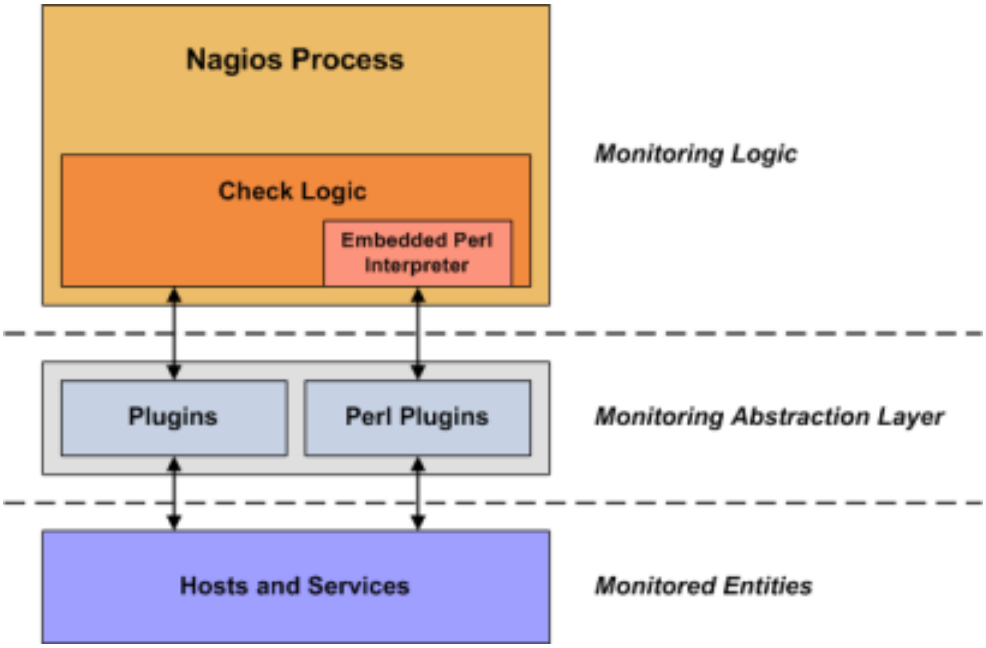

DEPRECATED IMAGE - TODO Replace with the Shinken specific architecture diagram.

Plugins act as an abstraction layer between the monitoring logic present in the Shinken daemon and the actual services and hosts that are being monitored.

The upside of this type of plugin architecture is that you can monitor just about anything you can think of. If you can automate the process of checking something, you can monitor it with Shinken. There are already literally thousands of plugins that have been created in order to monitor basic resources such as processor load, disk usage, ping rates, etc. If you want to monitor something else, take a look at the documentation on *writing plugins* and roll your own. It's simple!

The downside to this type of plugin architecture is the fact that Shinken has absolutely no idea what it is that you're monitoring. You could be monitoring network traffic statistics, data error rates, room temperate, CPU voltage, fan speed, processor load, disk space, or the ability of your super-fantastic toaster to properly brown your bread in the morning... Shinken doesn't understand the specifics of what's being monitored - it just tracks changes in the state of those resources. Only the plugins themselves know exactly what they're monitoring and how to perform the actual checks.

2.15 What Plugins Are Available?

There are plugins currently available to monitor many different kinds of devices and services.

They use basic monitoring protocols including:

- WMI, SNMP, SSH, NRPE, TCP, UDP, ICMP, OPC, LDAP and more

They can monitor pretty much anything:

- Unix/Linux, Windows, and Netware Servers
- Routers, Switches, VPNs
- Networked services: "HTTP", "POP3", "IMAP", "FTP", "SSH", "DHCP"

- CPU Load, Disk Usage, Memory Usage, Current Users
- Applications, databases, logs and more.

2.16 Obtaining Plugins

Shinken also organizes monitoring configuration packages. These are pre-built for fast no nonsense deployments. They include the check command definitions, service templates, host templates, discovery rules and integration hooks to the Community web site. The integration with the community web site permits for deployment and updates of monitoring packs.

Get started with Shinken Monitoring Packages "Packs" today.

The plugins themselves are not distributed with Shinken, but you can download the official Nagios plugins and many additional plugins created and maintained by Nagios users from the following locations:

- Nagios Plugins Project: http://nagiosplug.sourceforge.net/
- Nagios Downloads Page: http://www.nagios.org/download/
- NagiosExchange.org: http://www.nagiosexchange.org/

2.17 How Do I Use Plugin X?

Most plugins will display basic usage information when you execute them using "-h" or "–help" on the command line. For example, if you want to know how the **check_http** plugin works or what options it accepts, you should try executing the following command:

```
./check_http --help
```

2.18 Plugin API

You can find information on the technical aspects of plugins, as well as how to go about creating your own custom plugins *here*.

CHAPTER 3

Configuring Shinken

3.1 Configuration Overview

3.1.1 Introduction

There are several different configuration files that you're going to need to create or edit before you start monitoring anything. Be patient! Configuring Shinken can take quite a while, especially if you're first-time user. Once you figure out how things work, it'll all be well worth your time. :-)

Sample configuration files are installed in the "/usr/local/shinken/etc/" directory when you follow the *Quickstart installation guide*.

3.1.2 Main Configuration File

Main Shinken configuration files are separate in two part :

- Nagios compatible part (nagios.cfg)
- Shinken specific part (shinken-specific.cfg)

Documentation for the main configuration file can be found *Main Configuration File Options*.

3.1.3 Resource File(s)

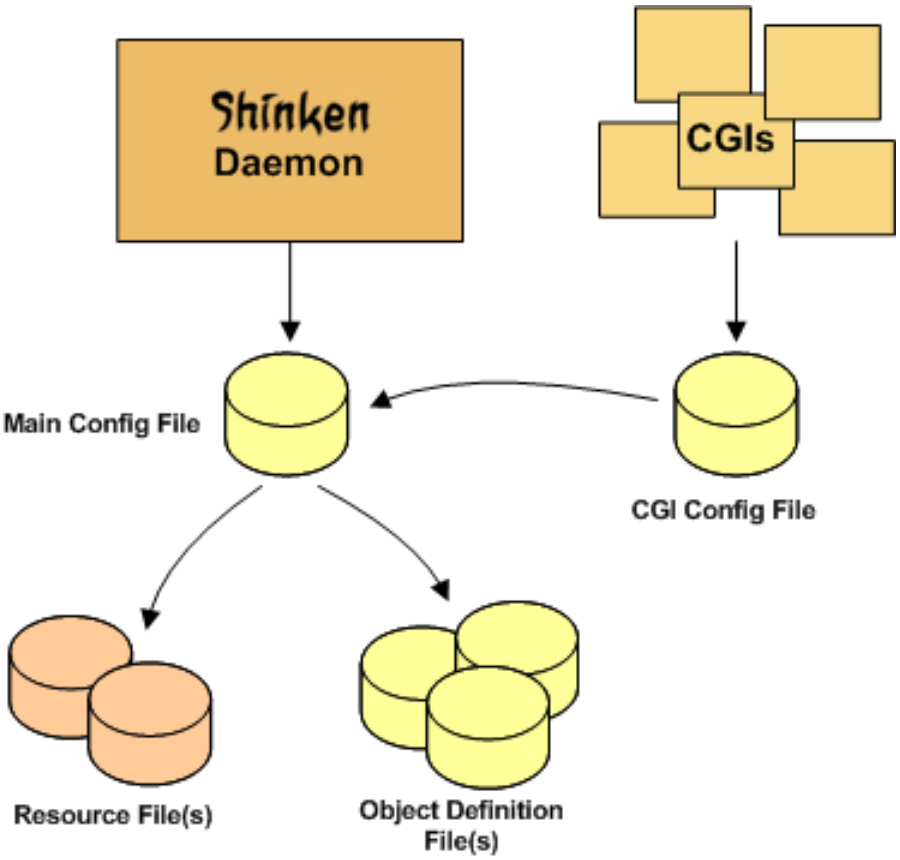

Resource files can be used to store user-defined macros. The main point of having resource files is to use them to store sensitive configuration information (like passwords), without making them available to the CGIs.

You can specify one or more optional resource files by using the *resource_file* directive in your main configuration file. ngshinken-configmain#configuringshinken-configmain-resource_file

3.1.4 Object Definition Files

Object definition files are used to define hosts, services, hostgroups, contacts, contactgroups, commands, etc. This is where you define all the things you want monitor and how you want to monitor them.

You can specify one or more object definition files by using the *cfg_file* and/or *cfg_dir* directives in your main configuration file.

An introduction to object definitions, and how they relate to each other, can be found *here*.

3.2 Main Configuration File Options

When creating and/or editing configuration files, keep the following in mind:

- Lines that start with a '"#"' character are taken to be comments and are not processed
- Variable names are case-sensitive
- If you want to configure a process to use a specific module:
  - You must define the module in a **modules_xxx.cfg** file in the **shinken-specific.d** directory
  - You must reference it in the **modules** section for that process, e.g. the **broker.cfg** file

3.2.1 Config File Location and sample

The main configuration files are usually named "nagios.cfg" and "shinken-specific*.cfg". They are located in the "/etc/shinken/" directory. Sample main configuration files ("/etc/shinken/nagios.cfg", "/etc/shiken/shinken-specific*.cfg") are installed for you when you follow the *Quickstart installation guide*.

3.2.2 Broker Modules

Shinken provides a lot of functionality through its interfaces with external systems. To this end, the broker daemon will load modules. The function of the modules is described in more detail in the *Broker modules page*.

Broker modules are essential for the web frontends, the metric databases, logging state changes in log management systems.

3.2.3 Arbiter, Receiver, Poller, Reactionner Modules

Shinken daemons can also load modules to influence what they can do and how they interface with external systems.

The sample configuration file provides succinct explanations of each module, the shinken architecture page also links to the different module configuration pages.

3.2.4 Path, users and log variables

Below you will find descriptions of each main Shinken configuration file option.

Log File

Defined in shinken-specific.cfg file.

Format

```
define broker{
     modules           <logging modules>
     [...]
}
```

Example for logging module named "Simple_log"

```
define module{
    module_name      Simple-log
    module_type      simple_log
    path             /var/lib/shinken/nagios.log
    archive_path     /var/lib/shinken/archives/
}
```

This variable specifies where Shinken should create its main log file **on the broker server**. If you have *Log Rotation Method* enabled, this file will automatically be rotated every day.

Log Level

Defined in nagios.cfg file.

Format

```
log_level=[DEBUG,INFO,WARNING,ERROR,CRITICAL]
```

Example :

```
log_level=WARNING
```

This variable specifies which logs will be raised by the arbiter daemon. For others daemons, it can be defined in their local *d.ini files.

Human format for log timestamp

Say if the timespam should be a unixtime (default) or a human read one.

Format :

```
human_timestamp_log=[0/1]
```

Example

```
human_timestamp_log=0
```

This directive is used to specify if the timespam before the log entry should be in unixtime (like [1302874960]) which is the default, or a human readable one (like [Fri Apr 15 15:43:19 2011]).

Beware : if you set the human format, some automatic parsing log tools won't work!

Object Configuration File

Defined in nagios.cfg file.

Format :

```
cfg_file=<file_name>
```

Example

```
cfg_file=/usr/local/shinken/etc/hosts.cfg
cfg_file=/usr/local/shinken/etc/services.cfg
cfg_file=/usr/local/shinken/etc/commands.cfg
```

This directive is used to specify an *Object Configuration Overview* containing object definitions that Shinken should use for monitoring. Object configuration files contain definitions for hosts, host groups, contacts, contact groups, services, commands, etc. You can seperate your configuration information into several files and specify multiple "cfg_file=" statements to have each of them processed.

Remark : the *cfg_file* can be a relative path, so it can be relative from the file that is reading. For example if you set "cfg_file=hosts.cfg" in the file "cfg_file=/etc/shinken/nagios.cfg", the file that will be read is "/etc/shinken/hosts.cfg".

Object Configuration Directory

Defined in nagios.cfg file.

Format:

```
cfg_dir=<directory_name>
```

Example:

```
cfg_dir=/etc/shinken/commands
cfg_dir=/etc/shinken/services
cfg_dir=/etc/shinken/hosts
```

This directive is used to specify a directory which contains *Object Configuration Overview* that Shinken should use for monitoring. All files in the directory with a .cfg extension are processed as object config files. Additionally, it will recursively process all config files in subdirectories of the directory you specify here. You can separate your configuration files into different directories and specify multiple

```
cfg_dir=
```

statements to have all config files in each directory processed.

Resource File

Defined in nagios.cfg file.

Format: resource_file=<file_name>

Example:

```
resource_file=/etc/shinken/resource.cfg
```

This is used to specify an optional resource file that can contain "$USERn$" *Understanding Macros and How They Work* definitions. "$USERn$" macros are useful for storing usernames, passwords, and items commonly used in command definitions (like directory paths). A classical variable used is $USER1$, used to store the plugins path, "/usr/local/nagios/libexec" on a classic installation.

Arbiter Daemon User

Defined in brokerd.ini, brokerd-windows.ini, pollerd.ini, pollerd-windows.ini, reactionnerd.ini, schedulerd.ini and schedulerd-windows.ini.

Format:

```
user=username
```

Example:

```
user=shinken
```

This is used to set the effective user that the **Arbiter** process (main process) should run as. After initial program startup, Shinken will drop its effective privileges and run as this user.

Arbiter Daemon user Group

Defined in brokerd.ini, brokerd-windows.ini, pollerd.ini, pollerd-windows.ini, reactionnerd.ini, schedulerd.ini and schedulerd-windows.ini.

Format:

```
group=groupname
```

Example:

```
group=shinken
```

This is used to set the effective group of the user used to launch the **arbiter** daemon.

Bypass security checks for the Arbiter daemon

Defined in brokerd.ini, brokerd-windows.ini, pollerd.ini, pollerd-windows.ini, reactionnerd.ini, schedulerd.ini and schedulerd-windows.ini.

Format:

```
idontcareaboutsecurity=<0/1>
```

Example:

```
idontcareaboutsecurity=0
```

This option determines whether or not Shinken will allow the Arbiter daemon to run under the root account. If this option is disabled, Shinken will bailout if the *nagios_user* or the *nagios_group* is configured with the root account.

The Shinken daemons do not need root right. Without a good reason do not run thems under this account!

- 0 = Be a responsible administrator
- 1 = Make crazy your security manager

Notifications Option

Format:

```
enable_notifications=<0/1>
```

Example:

```
enable_notifications=1
```

This option determines whether or not Shinken will send out *notifications*. If this option is disabled, Shinken will not send out notifications for any host or service.

Values are as follows:

- 0 = Disable notifications
- 1 = Enable notifications (default)

Log Rotation Method (Not fully implemented)

Format:

```
log_rotation_method=<n/h/d/w/m>
```

Example:

```
log_rotation_method=d
```

This is the rotation method that you would like Shinken to use for your log file on the **broker server**. Values are as follows:

- n = None (don't rotate the log - this is the default)
- h = Hourly (rotate the log at the top of each hour)
- d = Daily (rotate the log at midnight each day)
- w = Weekly (rotate the log at midnight on Saturday)
- m = Monthly (rotate the log at midnight on the last day of the month)

Tip: From now, only the d (Daily) parameter is managed.

External Command Check Option

Format:

```
check_external_commands=<0/1>
```

Example:

```
check_external_commands=1
```

This option determines whether or not Shinken will check the *External Command File* for commands that should be executed with the **arbiter daemon**. More information on external commands can be found *here*.

- 0 = Don't check external commands (default)
- 1 = Check external commands (default)

Note: FIX ME : Find the real default value

External Command File

Defined in nagios.cfg file.

Format:

```
command_file=<file_name>
```

Example:

```
command_file=/var/lib/shinken/rw/nagios.cmd
```

This is the file that Shinken will check for external commands to process with the **arbiter daemon**. The command CGI writes commands to this file. The external command file is implemented as a named pipe (FIFO), which is created when Nagios starts and removed when it shuts down. More information on external commands can be found *here*.

Tip: This external command file is not managed under Windows system. Please use others way to send commands like the LiveStatus module for example.

Arbiter Lock File

Defined in nagios.cfg file.

Format: lock_file=<file_name> Example: lock_file=/var/lib/shinken/arbiterd.pid

This option specifies the location of the lock file that Shinken **arbiter daemon** should create when it runs as a daemon (when started with the "-d" command line argument). This file contains the process id (PID) number of the running **arbiter** process.

State Retention Option (Not implemented)

Format:

```
retain_state_information=<0/1>
```

Example:

```
retain_state_information=1
```

This option determines whether or not Shinken will retain state information for hosts and services between program restarts. If

- 0 = Don't retain state information
- 1 = Retain state information (default)

Note: Idea to approve : Mark it as Unused : Related topic. A Shinken module replace it.

State Retention File

Format:

```
state_retention_file=<file_name>
```

Example:

```
state_retention_file=/var/lib/shinken/retention.dat
```

This is the file that Shinken **scheduler daemons** will use for storing status, downtime, and comment information before they shuts down. When Shinken is restarted it will use the information stored in this file for setting the initial states of services and hosts before it starts monitoring anything. In order to make Shinken retain state information between program restarts, you must enable the *State Retention Option* option.

Important: The file format is not the same between Shinken and Nagios! The retention.dat generated with Nagios will not load into Shinken.

Automatic State Retention Update Interval

Format:

```
retention_update_interval=<minutes>
```

Example:

```
retention_update_interval=60
```

This setting determines how often (in minutes) that Shinken **scheduler** will automatically save retention data during normal operation. If you set this value to 0, it will not save retention data at regular intervals, but it will still save retention data before shutting down or restarting. If you have disabled state retention (with the *State Retention Option* option), this option has no effect.

3.2.5 Scheduling parameters

Service/Host Check Execution Option

Format:

```
execute_service_checks=<0/1>
execute_host_checks=<0/1>
```

Example:

```
execute_service_checks=1
execute_host_checks=1
```

This option determines whether or not Shinken will execute service/host checks. Do not change this option unless you use a old school distributed architecture. And even if you do this, please change your architecture with a cool new one far more efficient.

- 0 = Don't execute service checks
- 1 = Execute service checks (default)

Passive Service/Host Check Acceptance Option

Format:

```
accept_passive_service_checks=<0/1>
accept_passive_host_checks=<0/1>
```

Example:

```
accept_passive_service_checks=1
accept_passive_host_checks=1
```

This option determines whether or not Shinken will accept *passive service/host checks*. If this option is disabled, Nagios will not accept any passive service/host checks.

- 0 = Don't accept passive service/host checks
- 1 = Accept passive service/host checks (default)

Event Handler Option

Format:

```
enable_event_handlers=<0/1>
```

Example:

```
enable_event_handlers=1
```

This option determines whether or not Shinken will run *event handlers*.

- 0 = Disable event handlers
- 1 = Enable event handlers (default)

Event Handler during downtimes

Format:

```
no_event_handlers_during_downtimes=<0/1>
```

Example:

```
no_event_handlers_during_downtimes=1
```

This option determines whether or not Shinken will run *event handlers* when the host or service is in a scheduled downtime.

- 0 = Disable event handlers (Nagios behavior) (default)
- 1 = Enable event handlers

References:

- http://www.mail-archive.com/shinken-devel@lists.sourceforge.net/msg01394.html
- https://github.com/naparuba/shinken/commit/9ce28d80857c137e5b915b39bbb8c1baecc821f9

Syslog Logging Option

Format:

```
use_syslog=<0/1>
```

Example:

```
use_syslog=1
```

This variable determines whether messages are logged to the syslog facility on your local host. Values are as follows:

- 0 = Don't use syslog facility
- 1 = Use syslog facility

Tip: This is a Unix Os only option.

Notification Logging Option

Format:

```
log_notifications=<0/1>
```

Example:

```
log_notifications=1
```

This variable determines whether or not notification messages are logged. If you have a lot of contacts or regular service failures your log file will grow (let say some Mo by day for a huge configuration, so it's quite OK for nearly every one to log them). Use this option to keep contact notifications from being logged.

- 0 = Don't log notifications
- 1 = Log notifications

Service/Host Check Retry Logging Option (Not implemented)

Format:

```
log_service_retries=<0/1>
log_host_retries=<0/1>
```

Example:

```
log_service_retries=0
log_host_retries=0
```

This variable determines whether or not service/host check retries are logged. Service check retries occur when a service check results in a non-OK state, but you have configured Shinken to retry the service more than once before responding to the error. Services in this situation are considered to be in "soft" states. Logging service check retries is mostly useful when attempting to debug Shinken or test out service/host *event handlers*.

- 0 = Don't log service/host check retries (default)
- 1 = Log service/host check retries

Event Handler Logging Option

Format:

```
log_event_handlers=<0/1>
```

Example:

```
log_event_handlers=1
```

This variable determines whether or not service and host *event handlers* are logged. Event handlers are optional commands that can be run whenever a service or hosts changes state. Logging event handlers is most useful when debugging Shinken or first trying out your event handler scripts.

- 0 = Don't log event handlers
- 1 = Log event handlers

Initial States Logging Option (Not implemented)

Format:

```
log_initial_states=<0/1>
```

Example:

```
log_initial_states=1
```

This variable determines whether or not Shinken will force all initial host and service states to be logged, even if they result in an OK state. Initial service and host states are normally only logged when there is a problem on the first check. Enabling this option is useful if you are using an application that scans the log file to determine long-term state statistics for services and hosts.

- 0 = Don't log initial states (default)
- 1 = Log initial states

External Command Logging Option

Format:

```
log_external_commands=<0/1>
```

Example:

```
log_external_commands=1
```

This variable determines whether or not Shinken will log *external commands* that it receives.

- 0 = Don't log external commands
- 1 = Log external commands (default)

Passive Check Logging Option (Not implemented)

Format:

```
log_passive_checks=<0/1>
```

Example:

```
log_passive_checks=1
```

This variable determines whether or not Shinken will log *passive host and service checks* that it receives from the *external command file*.

- 0 = Don't log passive checks
- 1 = Log passive checks (default)

Global Host/Service Event Handler Option (Not implemented)

Format:

```
global_host_event_handler=<command>
global_service_event_handler=<command>
```

Example:

```
global_host_event_handler=log-host-event-to-db
global_service_event_handler=log-service-event-to-db
```

This option allows you to specify a host event handler command that is to be run for every host state change. The global event handler is executed immediately prior to the event handler that you have optionally specified in each host definition. The command argument is the short name of a command that you define in your *Object Configuration Overview*. The maximum amount of time that this command can run is controlled by the *Event Handler Timeout* option. More information on event handlers can be found *here*.

Such commands should not be so useful with the new Shinken distributed architecture. If you use it, look if you can avoid it because such commands will kill your performances.

Maximum Host/Service Check Spread

Format:

```
max_service_check_spread=<minutes>
```

Example:

```
max_service_check_spread=30
```

This option determines the maximum number of minutes from when Shinken starts that all hosts/services (that are scheduled to be regularly checked) are checked. This option will ensure that the initial checks of all hosts/services occur within the timeframe you specify. Default value is 30 (minutes).

Timing Interval Length

Format:

```
interval_length=<seconds>
```

Example:

```
interval_length=60
```

This is the number of seconds per "unit interval" used for timing in the scheduling queue, re-notifications, etc. "Units intervals" are used in the object configuration file to determine how often to run a service check, how often to re-notify a contact, etc.

The default value for this is set to 60, which means that a "unit value" of 1 in the object configuration file will mean 60 seconds (1 minute).

Tip: Set this option top 1 is not a good thing with Shinken. It's not design to be a hard real time (<5seconds) monitoring system. Nearly no one need such hard real time (maybe only the Nuclear center or a market place like the London Exchange...).

3.2.6 Tuning and advanced parameters

Others parameters are useful for advanced features like flapping detection or performance tuning. Please look at the *configuringshinken-configmain-advanced* page for them.

3.2.7 Old CGI related parameter

If you are using the old CGI from Nagios, please migrate to a new WebUI. For historical perspective you can find information on the *specific CGI parameters*.

3.2.8 Unused parameters

The below parameters are inherited from Nagios but are not used in Shinken. You can defined them but if you don't it will be the same :)

They are listed on another page *Unused Nagios parameters*.

3.3 Object Configuration Overview

3.3.1 What Are Objects?

Objects are all the elements that are involved in the monitoring and notification logic. Types of objects include:

- Services
- Service Groups
- Hosts
- Host Groups
- Contacts
- Contact Groups
- Commands
- Time Periods
- Notification Escalations
- Notification and Execution Dependencies

More information on what objects are and how they relate to each other can be found below.

3.3.2 Where Are Objects Defined?

Objects can be defined in one or more configuration files and/or directories that you specify using the *cfg_file* and/or *cfg_dir* directives in the main configuration file.

When you follow the *Quickstart installation guide*, several sample object configuration files are placed in "/etc/shinken/objects/". You can use these sample files to see how object inheritance works and learn how to define your own object definitions.

3.3.3 How Are Objects Defined?

Objects are defined in a flexible template format, which can make it much easier to manage your Shinken configuration in the long term. Basic information on how to define objects in your configuration files can be found *here*.

Once you get familiar with the basics of how to define objects, you should read up on *object inheritance*, as it will make your configuration more robust for the future. Seasoned users can exploit some advanced features of object definitions as described in the documentation on *object tricks*.

3.3.4 Objects Explained

Some of the main object types are explained in greater detail below...

Hosts

Hosts are one of the central objects in the monitoring logic. Important attributes of hosts are as follows:

- Hosts are usually physical devices on your network (servers, workstations, routers, switches, printers, etc).
- Hosts have an address of some kind (e.g. an IP or MAC address).
- Hosts have one or more more services associated with them.
- Hosts can have parent/child relationships with other hosts, often representing real-world network connections, which is used in the *network reachability* logic.

Host Groups

***Host Groups* are groups of one or more hosts. Host groups can make it easier to**

- view the status of related hosts in the Shinken web interface and
- simplify your configuration through the use of *object tricks*.

Services

Services are one of the central objects in the monitoring logic. Services are associated with hosts and can be:

- Attributes of a host (CPU load, disk usage, uptime, etc.)
- Services provided by the host ("HTTP", "POP3", "FTP", "SSH", etc.)
- Other things associated with the host ("DNS" records, etc.)

Service Groups

***Service Groups* are groups of one or more services. Service groups can make it easier to**

- view the status of related services in the Shinken web interface and
- simplify your configuration through the use of *object tricks*.

Contacts

Contacts are people involved in the notification process:

- Contacts have one or more notification methods (cellphone, pager, email, instant messaging, etc.)
- Contacts receive notifications for hosts and service they are responsible for

Contact Groups

Contact Groups are groups of one or more contacts. Contact groups can make it easier to define all the people who get notified when certain host or service problems occur.

Timeperiods

Timeperiods are are used to control:

- When hosts and services can be monitored
- When contacts can receive notifications

Information on how timeperiods work can be found *here*.

Commands

Commands are used to tell Shinken what programs, scripts, etc. it should execute to perform:

- Host and service checks
- Notifications
- Event handlers
- and more...

3.4 Object Definitions

3.4.1 Introduction

One of the features of Shinken' object configuration format is that you can create object definitions that inherit properties from other object definitions. An explanation of how object inheritance works can be found *here*. I strongly suggest that you familiarize yourself with object inheritance once you read over the documentation presented below, as it will make the job of creating and maintaining object definitions much easier than it otherwise would be. Also, read up on the *object tricks* that offer shortcuts for otherwise tedious configuration tasks.

When creating and/or editing configuration files, keep the following in mind:

- Lines that start with a '"#"' character are taken to be comments and are not processed
- Directive names are case-sensitive

3.4.2 Sample Configuration Files

Sample object configuration files are installed in the "/etc/shinken/" directory when you follow the *quickstart installation guide*.

3.4.3 Object Types

- *Host*
- *Host Group*
- *Service*
- *Service Group*
- *Contact*
- *Contact Group*
- *Time Period*
- *Command*
- *Service Dependency*
- *Service Escalation*
- *Host Dependency*
- *Host Escalation*
- *Extended Host Information*
- *Extended Service Information*
- *Realm*
- *Arbiter*
- *Scheduler*
- *Poller*
- *Reactionner*
- *Broker*

3.5 Custom Object Variables

3.5.1 Introduction

Users often request that new variables be added to host, service, and contact definitions. These include variables for "SNMP" community, MAC address, AIM username, Skype number, and street address. The list is endless. The problem that I see with doing this is that it makes Nagios less generic and more infrastructure-specific. Nagios was intended to be flexible, which meant things needed to be designed in a generic manner. Host definitions in Nagios, for example, have a generic "address" variable that can contain anything from an IP address to human-readable driving directions - whatever is appropriate for the user's setup.

Still, there needs to be a method for admins to store information about their infrastructure components in their Nagios configuration without imposing a set of specific variables on others. Nagios attempts to solve this problem by allowing users to define custom variables in their object definitions. Custom variables allow users to define additional properties in their host, service, and contact definitions, and use their values in notifications, event handlers, and host and service checks.

3.5.2 Custom Variable Basics

There are a few important things that you should note about custom variables:

- Custom variable names must begin with an underscore (_) to prevent name collision with standard variables
- Custom variable names are case-insensitive
- Custom variables are *inherited* from object templates like normal variables
- Scripts can reference custom variable values with *macros and environment variables*

3.5.3 Examples

Here's an example of how custom variables can be defined in different types of object definitions:

```
define host{
   host_name       linuxserver
   _mac_address    00:06:5B:A6:AD:AA ; <-- Custom MAC_ADDRESS variable
   _rack_number    R32               ; <-- Custom RACK_NUMBER variable
...
}

define service{
   host_name        linuxserver
   description      Memory Usage
   _SNMP_community  public         ; <-- Custom SNMP_COMMUNITY variable
   _TechContact     Jane Doe       ; <-- Custom TECHCONTACT variable
   ....
}

define contact{
   contact_name    john
   _AIM_username   john16        ; <-- Custom AIM_USERNAME variable
   _YahooID        john32        ; <-- Custom YAHOOID variable
   ...
}
```

3.5.4 Custom Variables As Macros

Custom variable values can be referenced in scripts and executables that Nagios runs for checks, notifications, etc. by using *macros* or environment variables.

In order to prevent name collision among custom variables from different object types, Nagios prepends "_HOST", "_SERVICE", or "_CONTACT" to the beginning of custom host, service, or contact variables, respectively, in macro and environment variable names. The table below shows the corresponding macro and environment variable names for the custom variables that were defined in the example above.

Object Type	Variable Name	Macro Name	Environment Variable
Host	MAC_ADDRESS	$_HOSTMAC_ADDRESS$	NAGIOS__HOSTMAC_ADDRESS
Host	RACK_NUMBER	$_HOSTRACK_NUMBER$	NAGIOS__HOSTRACK_NUMBER
Service	SNMP_COMMUNITY	$_SERVICES-NMP_COMMUNITY$	NA-GIOS__SERVICESNMP_COMMUNITY
Service	TECHCONTACT	$_SERVICETECHCONTACT$	NAGIOS__SERVICETECHCONTACT
Contact	AIM_USERNAME	$_CONTAC-TAIM_USERNAME$	NA-GIOS__CONTACTAIM_USERNAME
Contact	YAHOOID	$_CONTACTYAHOOID$	NAGIOS__CONTACTYAHOOID

3.5.5 Custom Variables And Inheritance

Custom object variables are *inherited* just like standard host, service, or contact variables.

3.6 Main advanced configuration

3.6.1 Tuning and advanced parameters

Important: If you do not know how to change the values of theses parameters, don't touch them :) (and ask for help on the mailing list).

3.6.2 Performance data parameters

Performance Data Processor Command Timeout

Format:

```
perfdata_timeout=<seconds>
```

Example:

```
perfdata_timeout=5
```

This is the maximum number of seconds that Shinken will allow a *host performance data processor command* or *service performance data processor command* to be run. If a command exceeds this time limit it will be killed and a warning will be logged.

Performance Data Processing Option

Format:

```
process_performance_data=<0/1>
```

Example:

```
process_performance_data=1
```

This value determines whether or not Shinken will process host and service check *performance data*.

- 0 = Don't process performance data

- 1 = Process performance data (default)

If you want to use tools like PNP, NagiosGrapher or Graphite set it to 1.

Host/Service Performance Data Processing Command

Format:

```
host_perfdata_command=<command>
service_perfdata_command=<command>
```

Example:

```
host_perfdata_command=process-host-perfdata
services_perfdata_command=process-service-perfdata
```

This option allows you to specify a command to be run after every host/service check to process host/service *performance data* that may be returned from the check. The command argument is the short name of a *command definition* that you define in your object configuration file. This command is only executed if the *Performance Data Processing Option* option is enabled globally and if the "process_perf_data" directive in the *host definition* is enabled.

Host/Service Performance Data File

Format:

```
host_perfdata_file=<file_name>
service_perfdata_file=<file_name>
```

Example:

```
host_perfdata_file=/usr/local/shinken/var/host-perfdata.dat
service_perfdata_file=/usr/local/shinken/var/service-perfdata.dat
```

This option allows you to specify a file to which host/service *performance data* will be written after every host check. Data will be written to the performance file as specified by the *Host Performance Data File Template* option or the service one. Performance data is only written to this file if the *Performance Data Processing Option* option is enabled globally and if the "process_perf_data" directive in the *host definition* is enabled.

Host Performance Data File Template

Format:

```
host_perfdata_file_template=<template>
```

Example:

```
host_perfdata_file_template=[HOSTPERFDATA]\t$TIMET$\t$HOSTNAME$\t$HOSTEXECUTIONTIME$\t$HOSTOUTPUT$\t$
```

This option determines what (and how) data is written to the *host performance data file*. The template may contain *macros*, special characters (t for tab, r for carriage return, n for newline) and plain text. A newline is automatically added after each write to the performance data file.

Service Performance Data File Template

Format:

```
service_perfdata_file_template=<template>
```

Example:

```
service_perfdata_file_template=[SERVICEPERFDATA]\t$TIMET$\t$HOSTNAME$\t$SERVICEDESC$\t$$ERVICEEXECUT
```

This option determines what (and how) data is written to the *service performance data file*. The template may contain *macros*, special characters (t for tab, r for carriage return, n for newline) and plain text. A newline is automatically added after each write to the performance data file.

Host/Service Performance Data File Mode

Format:

```
host_perfdata_file_mode=<mode>
service_perfdata_file_mode=<mode>
```

Example:

```
host_perfdata_file_mode=a
service_perfdata_file_mode=a
```

This option determines how the *host performance data file* (or the service one) is opened. Unless the file is a named pipe you'll probably want to use the default mode of append.

- a = Open file in append mode (default)
- w = Open file in write mode
- p = Open in non-blocking read/write mode (useful when writing to pipes)

Host/Service Performance Data File Processing Interval (Unused)

Format:

```
host_perfdata_file_processing_interval=<seconds>
service_perfdata_file_processing_interval=<seconds>
```

Example:

```
host_perfdata_file_processing_interval=0
service_perfdata_file_processing_interval=0
```

This option allows you to specify the interval (in seconds) at which the *host performance data file* (orthe service one) is processed using the host performance data file processing command. A value of 0 indicates that the performance data file should not be processed at regular intervals.

Host/Service Performance Data File Processing Command (Unused)

Format:

```
host_perfdata_file_processing_command=<command>
service_perfdata_file_processing_command=<command>
```

Example:

```
host_perfdata_file_processing_command=process-host-perfdata-file
service_perfdata_file_processing_command=process-service-perfdata-file
```

This option allows you to specify the command that should be executed to process the *host performance data file* (or the service one). The command argument is the short name of a *command definition* that you define in your object configuration file. The interval at which this command is executed is determined by the *host_perfdata_file_processing_interval* directive.

3.6.3 Advanced scheduling parameters

Passive Host Checks Are SOFT Option (Not implemented)

Format:

```
passive_host_checks_are_soft=<0/1>
```

Example:

```
passive_host_checks_are_soft=1
```

This option determines whether or not Shinken will treat *passive host checks* as HARD states or SOFT states. By default, a passive host check result will put a host into a *HARD state type*. You can change this behavior by enabling this option.

- 0 = Passive host checks are HARD (default)
- 1 = Passive host checks are SOFT

Predictive Host/Service Dependency Checks Option (Unused)

Format:

```
enable_predictive_host_dependency_checks=<0/1>
enable_predictive_service_dependency_checks=<0/1>
```

Example:

```
enable_predictive_host_dependency_checks=1
enable_predictive_service_dependency_checks=1
```

This option determines whether or not Shinken will execute predictive checks of hosts/services that are being depended upon (as defined in *host/services dependencies*) for a particular host/service when it changes state. Predictive checks help ensure that the dependency logic is as accurate as possible. More information on how predictive checks work can be found *here*.

- 0 = Disable predictive checks
- 1 = Enable predictive checks (default)

Orphaned Host/Service Check Option

Format:

```
check_for_orphaned_services=<0/1>
check_for_orphaned_hosts=<0/1>
```

Example:

```
check_for_orphaned_services=1
check_for_orphaned_hosts=1
```

This option allows you to enable or disable checks for orphaned service/host checks. Orphaned checks are checks which have been launched to pollers but have not had any results reported in a long time.

Since no results have come back in for it, it is not rescheduled in the event queue. This can cause checks to stop being executed. Normally it is very rare for this to happen - it might happen if an external user or process killed off the process that was being used to execute a check.

If this option is enabled and Shinken finds that results for a particular check have not come back, it will log an error message and reschedule the check. If you start seeing checks that never seem to get rescheduled, enable this option and see if you notice any log messages about orphaned services.

- 0 = Don't check for orphaned service checks
- 1 = Check for orphaned service checks (default)

Max Plugins Output Length

Format:

```
max_plugins_output_length=<int>
```

Example:

```
max_plugins_output_length=8192
```

This option is used to set the max size in bytes for the checks plugins output. So if you saw truncated output like for huge disk check when you have a lot of partitions, raise this value.

Enable problem/impacts states change

Format:

```
enable_problem_impacts_states_change=<0/1>
```

Example:

```
enable_problem_impacts_states_change=0
```

This option is used to know if we apply or not the state change when an host or service is impacted by a root problem (like the service's host going down or a host's parent being down too). The state will be changed by UNKNONW for a service and UNREACHABLE for an host until their next schedule check. This state change do not count as a attempt, it's just for console so the users know that theses objects got problems and the previous states are not sure.

Soft State Dependencies Option (Not implemented)

Format: soft_state_dependencies=<0/1> Example: soft_state_dependencies=0

This option determines whether or not Shinken will use soft state information when checking *host and service dependencies*. Normally it will only use the latest hard host or service state when checking dependencies. If you want it to use the latest state (regardless of whether its a soft or hard *state type*), enable this option.

- 0 = Don't use soft state dependencies (default)
- 1 = Use soft state dependencies

3.6.4 Performance tuning

Cached Host/Service Check Horizon

Format:

```
cached_host_check_horizon=<seconds>
cached_service_check_horizon=<seconds>
```

Example:

```
cached_host_check_horizon=15
cached_service_check_horizon=15
```

This option determines the maximum amount of time (in seconds) that the state of a previous host check is considered current. Cached host states (from host/service checks that were performed more recently than the time specified by this value) can improve host check performance immensely. Too high of a value for this option may result in (temporarily) inaccurate host/service states, while a low value may result in a performance hit for host/service checks. Use a value of 0 if you want to disable host/service check caching. More information on cached checks can be found *here*.

Tip: Nagios default is 15s, but it's a tweak that make checks less accurate. So Shinken use 0s as a default. If you have performances problems and you can't add a new scheduler or poller, increase this value and start to buy a new server because this won't be magical.

Large Installation Tweaks Option

Format:

```
use_large_installation_tweaks=<0/1>
```

Example:

```
use_large_installation_tweaks=0
```

This option determines whether or not the Shinken daemon will take shortcuts to improve performance. These shortcuts result in the loss of a few features, but larger installations will likely see a lot of benefit from doing so. If you can't add new satellites to manage the load (like new pollers), you can activate it. More information on what optimizations are taken when you enable this option can be found *here*.

- 0 = Don't use tweaks (default)
- 1 = Use tweaks

Environment Macros Option

Format:

```
enable_environment_macros=<0/1>
```

Example:

```
enable_environment_macros=0
```

This option determines whether or not the Shinken daemon will make all standard *macros* available as environment variables to your check, notification, event hander, etc. commands. In large installations this can be problematic because it takes additional CPU to compute the values of all macros and make them available to the environment. It also cost a increase network communication between schedulers and pollers.

- 0 = Don't make macros available as environment variables
- 1 = Make macros available as environment variables (default)

3.6.5 Flapping parameters

Flap Detection Option

Format:

```
enable_flap_detection=<0/1>
```

Example:

```
enable_flap_detection=1
```

This option determines whether or not Shinken will try and detect hosts and services that are "flapping". Flapping occurs when a host or service changes between states too frequently, resulting in a barrage of notifications being sent out. When Shinken detects that a host or service is flapping, it will temporarily suppress notifications for that host/service until it stops flapping.

More information on how flap detection and handling works can be found *here*.

- 0 = Don't enable flap detection (default)
- 1 = Enable flap detection

Low Service/Host Flap Threshold

Format:

```
low_service_flap_threshold=<percent>
low_host_flap_threshold=<percent>
```

Example:

```
low_service_flap_threshold=25.0
low_host_flap_threshold=25.0
```

This option is used to set the low threshold for detection of host/service flapping. For more information on how flap detection and handling works (and how this option affects things) read *this*.

High Service/Host Flap Threshold

Format:

```
high_service_flap_threshold=<percent>
high_host_flap_threshold=<percent>
```

Example:

```
high_service_flap_threshold=50.0
high_host_flap_threshold=50.0
```

This option is used to set the high threshold for detection of host/service flapping. For more information on how flap detection and handling works (and how this option affects things) read *this*.

Flap History

Format:

```
flap_history=<int>
```

Example:

```
flap_history=20
```

This option is used to set the history size of states keep by the scheduler to make the flapping calculation. By default, the value is 20 states kept.

The size in memory is for the scheduler daemon : 4Bytes * flap_history * (nb hosts + nb services). For a big environment, it costs 4 * 20 * (1000+10000) ~ 900Ko. So you can raise it to higher value if you want. To have more information about flapping, you can read *this*.

3.6.6 Commands/checks timeout

Service/Host Check Timeout

Format:

```
service_check_timeout=<seconds>
```

Example:

```
service_check_timeout=60
```

This is the maximum number of seconds that Shinken will allow service checks to run. If checks exceed this limit, they are killed and a CRITICAL state is returned. A timeout error will also be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off plugins which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more), so that each check normally finishes executing within this time limit. If a check runs longer than this limit, Shinken will kill it off thinking it is a runaway processes.

Various commands Timeouts

Format:

```
event_handler_timeout=<seconds>  # default: 30s
notification_timeout=<seconds>   # default: 30s
ocsp_timeout=<seconds>           # default: 15s
ochp_timeout=<seconds>           # default: 15s
```

Example:

```
event_handler_timeout=60
notification_timeout=60
ocsp_timeout=5
ochp_timeout=5
```

This is the maximum number of seconds that Shinken will allow *event handlers*, notification, *obsessive compulsive service processor command* or a *Obsessive Compulsive Host Processor Command* to be run. If an command exceeds this time limit it will be killed and a warning will be logged.

There is often widespread confusion as to what this option really does. It is meant to be used as a last ditch mechanism to kill off commands which are misbehaving and not exiting in a timely manner. It should be set to something high (like 60 seconds or more for notification, less for oc*p commands), so that each event handler command normally finishes executing within this time limit. If an event handler runs longer than this limit, Shinken will kill it off thinking it is a runaway processes.

3.6.7 Old Obsess Over commands

Obsess Over Services Option

Format:

```
obsess_over_services=<0/1>
```

Example:

```
obsess_over_services=1
```

This value determines whether or not Shinken will "obsess" over service checks results and run the *obsessive compulsive service processor command* you define. I know - funny name, but it was all I could think of. This option is useful for performing *distributed monitoring*. If you're not doing distributed monitoring, don't enable this option.

- 0 = Don't obsess over services (default)
- 1 = Obsess over services

Obsessive Compulsive Service Processor Command

Format:

```
ocsp_command=<command>
```

Example:

```
ocsp_command=obsessive_service_handler
```

This option allows you to specify a command to be run after every service check, which can be useful in *distributed monitoring*. This command is executed after any *event handler* or *notification* commands. The command argument is the short name of a *command definition* that you define in your object configuration file.

It's used nearly only for the old school distributed architecture. If you use it, please look at new architecture capabilities that are far efficient than the old one. More information on distributed monitoring can be found *here*. This command is only executed if the *Obsess Over Services Option* option is enabled globally and if the "obsess_over_service" directive in the *service definition* is enabled.

Obsess Over Hosts Option

Format:

```
obsess_over_hosts=<0/1>
```

Example:

```
obsess_over_hosts=1
```

This value determines whether or not Shinken will "obsess" over host checks results and run the *obsessive compulsive host processor command* you define. Same like the service one but for hosts :)

- 0 = Don't obsess over hosts (default)
- 1 = Obsess over hosts

Obsessive Compulsive Host Processor Command

Format:

```
ochp_command=<command>
```

Example:

```
ochp_command=obsessive_host_handler
```

This option allows you to specify a command to be run after every host check, which can be useful in *distributed monitoring*. This command is executed after any *event handler* or *notification* commands. The command argument is the short name of a *command definition* that you define in your object configuration file.

This command is only executed if the *Obsess Over Hosts Option* option is enabled globally and if the "obsess_over_host" directive in the *host definition* is enabled.

3.6.8 Freshness check

Host/Service Freshness Checking Option

Format:

```
check_service_freshness=<0/1>
check_host_freshness=<0/1>
```

Example:

```
check_service_freshness=0
check_host_freshness=0
```

This option determines whether or not Shinken will periodically check the "freshness" of host/service checks. Enabling this option is useful for helping to ensure that *passive service checks* are received in a timely manner. More information on freshness checking can be found *here*.

- 0 = Don't check host/service freshness
- 1 = Check host/service freshness (default)

Host/Service Freshness Check Interval

Format:

```
service_freshness_check_interval=<seconds>
host_freshness_check_interval=<seconds>
```

Example:

```
service_freshness_check_interval=60
host_freshness_check_interval=60
```

This setting determines how often (in seconds) Shinken will periodically check the "freshness" of host/service check results. If you have disabled host/service freshness checking (with the *check_service_freshness* option), this option has no effect. More information on freshness checking can be found *here*.

Additional Freshness Threshold Latency Option (Not implemented)

Format:

```
additional_freshness_latency=<#>
```

Example:

```
additional_freshness_latency=15
```

This option determines the number of seconds Shinken will add to any host or services freshness threshold it automatically calculates (e.g. those not specified explicitly by the user). More information on freshness checking can be found *here*.

3.6.9 All the others :)

Date Format (Not implemented)

Format:

```
date_format=<option>
```

Example:

```
date_format=us
```

This option allows you to specify what kind of date/time format Shinken should use in date/time *macros*. Possible options (along with example output) include:

Option	Output Format	Sample Output
us	MM/DD/YYYY HH:MM:SS	06/30/2002 03:15:00
euro	DD/MM/YYYY HH:MM:SS	30/06/2002 03:15:00
iso8601	YYYY-MM-DD HH:MM:SS	2002-06-30 03:15:00
strict-iso8601	YYYY-MM-DDTHH:MM:SS	2002-06-30T03:15:00

Timezone Option

Format:

```
use_timezone=<tz>
```

Example:

```
use_timezone=US/Mountain
```

This option allows you to override the default timezone that this instance of Shinken runs in. Useful if you have multiple instances of Shinken that need to run from the same server, but have different local times associated with them. If not specified, Shinken will use the system configured timezone.

Illegal Object Name Characters

Format:

```
illegal_object_name_chars=<chars...>
```

Example:

```
illegal_object_name_chars=`~!$%^&*"|'<>?,()=
```

This option allows you to specify illegal characters that cannot be used in host names, service descriptions, or names of other object types. Shinken will allow you to use most characters in object definitions, but I recommend not using the characters shown in the example above. Doing may give you problems in the web interface, notification commands, etc.

Illegal Macro Output Characters

Format:

```
illegal_macro_output_chars=<chars...>
```

Example:

```
illegal_macro_output_chars=`~$^&"|'<>
```

This option allows you to specify illegal characters that should be stripped from *macros* before being used in notifications, event handlers, and other commands. This DOES NOT affect macros used in service or host check commands. You can choose to not strip out the characters shown in the example above, but I recommend you do not do this. Some of these characters are interpreted by the shell (i.e. the backtick) and can lead to security problems. The following macros are stripped of the characters you specify:

- "$HOSTOUTPUT$"
- "$HOSTPERFDATA$"
- "$HOSTACKAUTHOR$"
- "$HOSTACKCOMMENT$"
- "$SERVICEOUTPUT$"
- "$SERVICEPERFDATA$"
- "$SERVICEACKAUTHOR$"
- "$SERVICEACKCOMMENT$"

Regular Expression Matching Option (Not implemented)

Format:

```
use_regexp_matching=<0/1>
```

Example:

```
use_regexp_matching=0
```

This option determines whether or not various directives in your *Object Configuration Overview* will be processed as regular expressions. More information on how this works can be found *here*.

- 0 = Don't use regular expression matching (default)
- 1 = Use regular expression matching

True Regular Expression Matching Option (Not implemented)

Format:

```
use_true_regexp_matching=<0/1>
```

Example:

```
use_true_regexp_matching=0
```

If you've enabled regular expression matching of various object directives using the *Regular Expression Matching Option* option, this option will determine when object directives are treated as regular expressions. If this option is disabled (the default), directives will only be treated as regular expressions if they contain *, ?, +, or .. If this option is enabled, all appropriate directives will be treated as regular expression - be careful when enabling this! More information on how this works can be found *here*.

- 0 = Don't use true regular expression matching (default)
- 1 = Use true regular expression matching

Administrator Email Address (unused)

Format:

```
admin_email=<email_address>
```

Example:

```
admin_email=root@localhost.localdomain
```

This is the email address for the administrator of the local machine (i.e. the one that Shinken is running on). This value can be used in notification commands by using the "$ADMINEMAIL$" *macro*.

Administrator Pager (unused)

Format:

```
admin_pager=<pager_number_or_pager_email_gateway>
```

Example:

```
admin_pager=pageroot@localhost.localdomain
```

This is the pager number (or pager email gateway) for the administrator of the local machine (i.e. the one that Shinken is running on). The pager number/address can be used in notification commands by using the $ADMINPAGER$ *macro*.

CHAPTER 4

Running Shinken

4.1 Verifying Your Configuration

Every time you modify your *Configuration Overview*, you should run a sanity check on them. It is important to do this before you (re)start Shinken, as Shinken will shut down if your configuration contains errors.

4.1.1 How to verify the configuration

In order to verify your configuration, run Shinken-arbiter with the "-v" command line option like so:

```
linux:~ # /usr/local/shinken/bin/shinken-arbiter -v -c /usr/local/shinken/etc/nagios.cfg -c /usr/loca
```

If you've forgotten to enter some critical data or misconfigured things, Shinken will spit out a warning or error message that should point you to the location of the problem. Error messages generally print out the line in the configuration file that seems to be the source of the problem. On errors, Shinken will exit the pre-flight check. If you get any error messages you'll need to go and edit your configuration files to remedy the problem. Warning messages can generally be safely ignored, since they are only recommendations and not requirements.

4.1.2 Important caveats

1. Shinken will not check the syntax of module variables
2. Shinken will not check the validity of data passed to modules
3. Shinken will NOT notify you if you mistyped an expected variable, it will treat it as a custom variable.
4. Shinken sometimes uses variables that expect lists, the order of items in lists is important, check the relevant documentation

4.1.3 How to apply your changes

Once you've verified your configuration files and fixed any errors you can go ahead and *(re)start Shinken*.

4.2 Starting and Stopping Shinken

There's more than one way to start, stop, and restart Shinken. Here are some of the more common ones...

Always make sure you *Verifying Your Configuration* before you (re)start Shinken.

4.2.1 Starting Shinken

- Init Script: The easiest way to start the Shinken daemon is by using the init script like so:

```
linux:~ # /etc/rc.d/init.d/shinken start
```

- Manually: You can start the Shinken daemon manually with the "-d" command line option like so:

```
linux:~ # /usr/local/shinken/bin/shinken-scheduler -d -c /usr/local/shinken/etc/schedulerd.ini
linux:~ # /usr/local/shinken/bin/shinken-poller -d -c /usr/local/shinken/etc/pollerd.ini
linux:~ # /usr/local/shinken/bin/shinken-reactionner -d -c /usr/local/shinken/etc/reactionnerd.ini
linux:~ # /usr/local/shinken/bin/shinken-broker -d -c /usr/local/shinken/etc/brokerd.ini
linux:~ # /usr/local/shinken/bin/shinken-arbiter -d -c /usr/local/shinken/etc/nagios.cfg -c /usr/loca
```

Important: Enabling debugging output under windows requires changing registry values associated with Shinken

4.2.2 Restarting Shinken

Restarting/reloading is nececessary when you modify your configuration files and want those changes to take effect.

- Init Script: The easiest way to restart the Shinken daemon is by using the init script like so:

```
linux:~ # /etc/rc.d/init.d/shinken restart
```

- Manually: You can restart the Shinken process by sending it a SIGTERM signal like so:

```
linux:~ # kill <arbiter_pid>
linux:~ # /usr/local/shinken/bin/shinken-arbiter. -d -c /usr/local/shinken/etc/nagios.cfg
```

4.2.3 Stopping Shinken

- Init Script: The easiest way to stop the Shinken daemons is by using the init script like so:

```
linux:~ # /etc/rc.d/init.d/shinken stop
```

- Manually: You can stop the Shinken process by sending it a SIGTERM signal like so:

```
linux:~ # kill <arbiter_pid> <scheduler_pid> <poller_pid> <reactionner_pid> <broker_pid>
```

CHAPTER 5

The Basics

5.1 Setting up a basic Shinken Configuration

5.1.1 Default Shinken configuration

If you followed the *10 Minute Shinken Installation Guide* tutorial you were able to install and launch Shinken.

The default configuration deployed with the Shinken sources contains:

- one arbiter
- one scheduler
- one poller
- one reactionner
- one broker
- one receiver (commented out)

All these elements must have a basic configuration. The Arbiter must know about the other daemons and how to communicate with them, just as the other daemons need to know on which TCP port they must listen on.

5.1.2 Configure the Shinken Daemons

The schedulers, pollers, reactionners and brokers daemons need to know in which directory to work on, and on which TCP port to listen. That's all.

Note: If you plan on using the default directories, user (shinken) and tcp port you shouldn't have to edit these files.

Each daemon has one configuration file. The default location is /usr/local/shinken/etc/.

Important: Remember that all daemons can be on different servers: the daemons configuration files need to be on the server which is running the daemon, not necessarily on every server

Let's see what it looks like:

```
$cat etc/schedulerd.ini

[daemon]
workdir=/usr/local/shinken/var
pidfile=%(workdir)s/schedulerd.pid
port=7768
host=0.0.0.0

daemon_enabled=1

# Optional configurations

user=shinken
group=shinken
idontcareaboutsecurity=0
use_ssl=0
#certs_dir=etc/certs
#ca_cert=etc/certs/ca.pem
#server_cert=etc/certs/server.pem
hard_ssl_name_check=0
use_local_log=1
```

```
local_log=brokerd.log
log_level=INFO
max_queue_size=100000
```

So here we have a scheduler:

- workdir: working directory of the daemon. By default /usr/local/shinken/var
- pidfile: pid file of the daemon (so we can kill it :)). By default /usr/local/shinken/var/schedulerd.pid for a scheduler.
- port: TCP port to listen to. By default:
  - scheduler: 7768
  - poller: 7771
  - reactionner: 7769
  - broker: 7772
  - arbiter: 7770 (the arbiter configuration will be seen later)
- host: IP interface to listen on. The default 0.0.0.0 means all interfaces
- user: user used by the daemon to run. By default shinken
- group: group of the user. By default shinken.
- idontcareaboutsecurity: if set to 1, you can run it under the root account. But seriously: do not to this. The default is 0 of course.
- daemon_enabled : if set to 0, the daemon won't run. Useful for distributed setups where you only need a poller for example.
- use_ssl=0
- #certs_dir=etc/certs
- #ca_cert=etc/certs/ca.pem
- #server_cert=etc/certs/server.pem
- hard_ssl_name_check=0
- use_local_log=1 : Log all messages that match the log_level for this daemon in a local directory
- local_log=brokerd.log : name of the log file where to save the logs
- log_level=INFO : Log_level that will be permitted to be logger. Warning permits Warning, Error, Critical to be logged. INFO by default.
- max_queue_size=100000 : If a module got a brok queue() higher than this value, it will be killed and restarted. Put to 0 to disable it

5.1.3 Daemon declaration in the global configuration

Now each daemon knows in which directory to run, and on which tcp port to listen. A daemon is a resource in the Shinken architecture. Such resources must be declared in the global configuration (where the Arbiter is) for them to be utilized.

The global configuration file is: **/usr/local/shinken/etc/shinken-specific.cfg/**

The daemon declarations are quite simple: each daemon is represented by an object. The information contained in the daemon object are network parameters about how its resources should be treated (is it a spare, ...).

Each objects type corresponds to a daemon:

- arbiter
- scheduler
- poller
- reactionner
- broker
- receiver

The names were chosen to understand their roles more easily. :)

They have these parameters in common:

- *_name: name of the resource
- address: IP or DNS address to connect to the daemon
- port: I think you can find it on your own by now :)
- [spare]: 1 or 0, is a spare or not. *See advanced features for this*.
- [realm]: realm membership *See advanced features for this*.
- [manage_sub_realms]: manage or not sub realms. *See advanced features for this*.
- [modules]: modules used by the daemon. See below.

special parameters

Some daemons have special parameters:

For the arbiter:

- host_name: hostname of the server where the arbiter is installed. It's mandatory for a high availability environment (2 arbiters or more).

For pollers:

- poller_tags: "tags" that the poller manages. See advanced features for this.

module objects

All daemons can use modules. In the brokers case, they are mandatory for it to actually accomplish a task.

Modules have some common properties:

- module_name: module name called by the resource.
- module_type: module type of the module. It's a fixed value given by the module.
- other options: each module can have specific parameters. See the respective module documentation to learn more about them.

Module references, *list of overall modules*:

- Arbiter modules
- *Scheduler modules*
- *Broker modules*

- Receiver modules
- Pollers modules
- Reactionner modules

Configuration example

Here is an example of a simple configuration (which you already used without knowing it during the 10min installation tutorial). It has been kept to the strict minimum, with only one daemon for each type. There is no load distribution or high availability, but you'll get the picture more easily.

Here, we have a server named server-1 that has 192.168.0.1 as its IP address:

```
define arbiter{
     arbiter_name  arbiter-1
     host_name     server-1
     address       192.168.0.1
     port          7770
     spare         0
}

define scheduler{
     scheduler_name   scheduler-1
     address          192.168.0.1
     port             7768
     spare            0
}

define reactionner{
     reactionner_name     reactionner-1
     address              192.168.0.1
     port                 7769
     spare                0
}

define poller{
     poller_name     poller-1
     address         192.168.0.1
     port            7771
     spare           0
}

define broker{
     broker_name      broker-1
     address          192.168.0.1
     port             7772
     spare            0
     modules          Status-Dat,Simple-log
}

define module{
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/shinken.log
}

define module{
     module_name              Status-Dat
```

```
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

See? That was easy. And don't worry about forgetting one of them: if there is a missing daemon type, Shinken automatically adds one locally with a default address/port configuration.

Removing unused configurations

The sample shinken-specific.cfg file has all possible modules in addition to the basic daemon declarations.

- Backup your shinken-specific.cfg file.
- Delete all unused modules from your configuration file - Ex. If you do not use the openldap module, delete it from the file

This will make any warnings or errors that show up in your log files more pertinent. This is because the modules, if declared will get loadedup even if they are not use in your Modules declaration of your daemons.

If you ever lose your shinken-specific.cfg, you can simply go to the shinken github repository and download the file.

launch all daemons

To launch daemons, simply type:

```
daemon_path -d -c daemon_configuration.ini
```

The command lines arguments are:

- -c, –config: Config file.
- -d, –daemon: Run in daemon mode
- -r, –replace: Replace previous running scheduler
- -h, –help: Print detailed help screen
- –debug: path of the debug file

So a standard launch of the resources looks like:

```
/usr/local/shinken/bin/shinken-scheduler -d -c /usr/local/shinken/etc/schedulerd.ini
/usr/local/shinken/bin/shinken-poller -d -c /usr/local/shinken/etc/pollerd.ini
/usr/local/shinken/bin/shinken-reactionner -d -c /usr/local/shinken/etc/reactionnerd.ini
/usr/local/shinken/bin/shinken-broker -d -c /usr/local/shinken/etc/brokerd.ini
```

Now we can start the arbiter with the global configuration:

```
#First we should check the configuration for errors
python bin/shinken-arbiter -v -c etc/nagios.cfg -c etc/shinken-specific.cfg

#then, we can really launch it
python bin/shinken-arbiter -d -c etc/nagios.cfg -c etc/shinken-specific.cfg
```

Now, you've got the same thing you had when you launched bin/launch_all.sh script 8-) (but now you know what you're doing)

5.1.4 What next

You are ready to continue to the next section, get DATA IN Shinken.

If you feel in the mood for testing even more shinken features, now would be the time to look at *advanced_features* to play with distributed and high availability architectures!

5.2 Nagios/Shinken Plugins

5.2.1 Introduction

Shinken includes a set of scalable internal mechanisms for checking the status of hosts and services on your network. These are called modules and can be loaded by the various Shinken daemons involved in data acquisition (Poller daemons, Receiver daemons, Arbiter Daemon) Shinken also relies on external programs (called plugins) to do monitor a very wide variety of devices, applications and networked services.

5.2.2 What Are Plugins?

Plugins are compiled executables or scripts (Perl scripts, shell scripts, etc.) that can be run from a command line to check the status or a host or service. Shinken uses the results from plugins to determine the current status of hosts and services on your network.

Shinken will execute a plugin whenever there is a need to check the status of a service or host. The plugin does something (notice the very general term) to perform the check and then simply returns the results to Shinken. It will process the results that it receives from the plugin and take any necessary actions (running *event handlers*, sending out *notifications*, etc).

5.2.3 Shinken integrated data acquisition modules

These replace traditional unscalable plugins with high performance variants that are more tightly coupled with Shinken.

- The Shinken architecture describes how to fit with the Shinken daemons
- The Shinken NRPE module configuration page describes how it works and its configuration

Integrated Shinken data acquisition modules support the following protocols:

- *NRPE*
- *SNMP*

5.2.4 Plugins As An Abstraction Layer

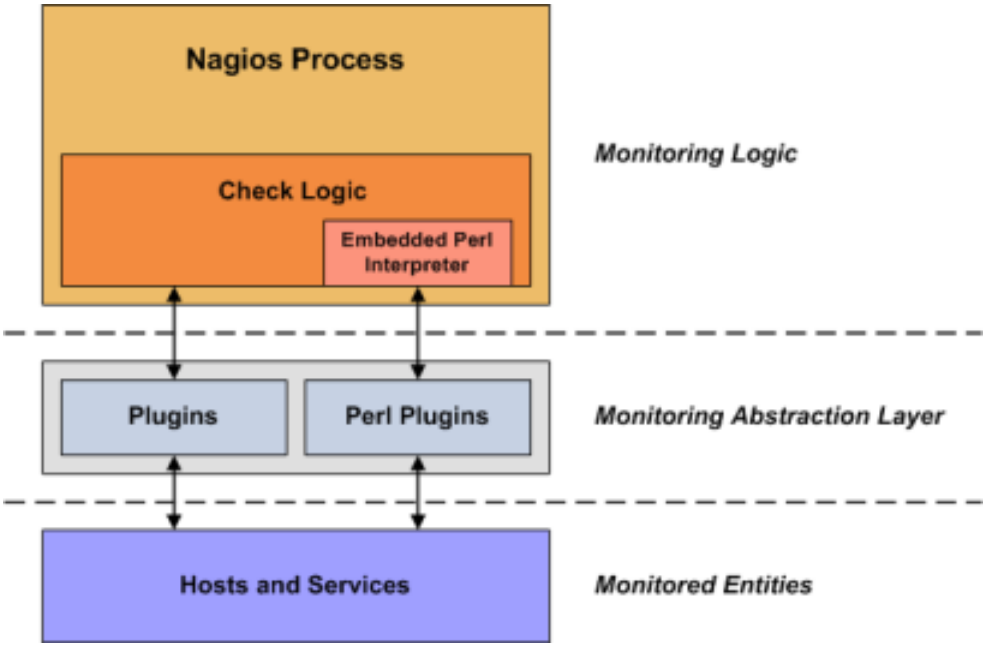

DEPRECATED IMAGE - TODO Replace with Shinken specific.

Plugins act as an abstraction layer between the monitoring logic present in the Shinken daemon and the actual services and hosts that are being monitored.

The upside of this type of plugin architecture is that you can monitor just about anything you can think of. If you can automate the process of checking something, you can monitor it with Shinken. There are already a lot of plugins that have been created in order to monitor basic resources such as processor load, disk usage, ping rates, etc. If you want to monitor something else, take a look at the documentation on *writing plugins* and roll your own. Its simple!

The downside to this type of plugin architecture is the fact that Shinken has absolutely no idea what it is that you're monitoring. You could be monitoring network traffic statistics, data error rates, room temperate, CPU voltage, fan speed, processor load, disk space, or the ability of your super-fantastic toaster to properly brown your bread in the morning... Shinken doesn't understand the specifics of what's being monitored - it just tracks changes in the state of those resources. Only the plugins themselves know exactly what they're monitoring and how to perform the actual checks.

5.2.5 What Plugins Are Available?

There are plugins currently available to monitor many different kinds of devices and services, including:

- "HTTP", "POP3", "IMAP", "FTP", "SSH", "DHCP"
- CPU Load, Disk Usage, Memory Usage, Current Users
- Unix/Linux, Windows, and Netware Servers
- Routers and Switches
- etc.

5.2.6 Obtaining Plugins

Shinken also organizes monitoring configuration packages. These are pre-built for fast no nonsense deployments. They include the check command definitions, service templates, host templates, discovery rules and integration hooks to the Community web site. The integration with the community web site permits for deployment and updates of monitoring packs.

Get started with Shinken Monitoring Packages "Packs" today.

The plugins themseles are not distributed with Shinken, but you can download the official Nagios plugins and many additional plugins created and maintained by Nagios users from the following locations:

- Nagios Plugins Project: http://nagiosplug.sourceforge.net/
- Nagios Downloads Page: http://www.nagios.org/download/
- NagiosExchange.org: http://www.nagiosexchange.org/

5.2.7 How Do I Use Plugin X?

Most all plugins will display basic usage information when you execute them using "-h" or "–help" on the command line. For example, if you want to know how the **check_http** plugin works or what options it accepts, you should try executing the following command:

```
./check_http --help
```

5.2.8 Plugin API

You can find information on the technical aspects of plugins, as well as how to go about creating your own custom plugins *here*.

5.3 Understanding Macros and How They Work

5.3.1 Macros

One of the main features that make Shinken so flexible is the ability to use macros in command defintions. Macros allow you to reference information from hosts, services, and other sources in your commands.

5.3.2 Macro Substitution - How Macros Work

Before Shinken executes a command, it will replace any macros it finds in the command definition with their corresponding values. This macro substitution occurs for all types of commands that Shinken executes - host and service checks, notifications, event handlers, etc.

Certain macros may themselves contain other macros. These include the "$HOSTNOTES$", "$HOSTNOTESURL$", "$HOSTACTIONURL$", "$SERVICENOTES$", "$SERVICENOTESURL$", and "$SERVICEACTIONURL$" macros.

Tip: If, you need to have the '$' character in one of your command (and not referring to a macro), please put "$$" instead. Shinken will replace it well

5.3.3 Example 1: Host Address Macro

When you use host and service macros in command definitions, they refer to values for the host or service for which the command is being run. Let's try an example. Assuming we are using a host definition and a check_ping command defined like this:

```
define host{
  host_name    linuxbox
  address    192.168.1.2
  check_command    check_ping
  ...
}

define command{
  command_name    check_ping
  command_line    /usr/local/shinken/libexec/check_ping -H $HOSTADDRESS$ -w 100.0,90% -c 200.0,60%
}
```

the expanded/final command line to be executed for the host's check command would look like this:

```
/usr/local/shinken/libexec/check_ping -H 192.168.1.2 -w 100.0,90% -c 200.0,60%
```

Pretty simple, right? The beauty in this is that you can use a single command definition to check an unlimited number of hosts. Each host can be checked with the same command definition because each host's address is automatically substituted in the command line before execution.

5.3.4 Example 2: Command Argument Macros

You can pass arguments to commands as well, which is quite handy if you'd like to keep your command definitions rather generic. Arguments are specified in the object (i.e. host or service) definition, by separating them from the command name with exclamation points (!) like so:

```
define service{
  host_name    linuxbox
  service_description    PING
  check_command    check_ping!200.0,80%!400.0,40%
  ...
}
```

In the example above, the service check command has two arguments (which can be referenced with *$ARGn$* macros). The $ARG1$ macro will be "200.0,80%" and "$ARG2$" will be "400.0,40%" (both without quotes). Assuming we are using the host definition given earlier and a **check_ping** command defined like this:

```
define command{
  command_name    check_ping
  command_line    /usr/local/shinken/libexec/check_ping -H $HOSTADDRESS$ -w $ARG1$ -c $ARG2$
}
```

the expanded/final command line to be executed for the service's check command would look like this:

```
/usr/local/shinken/libexec/check_ping -H 192.168.1.2 -w 200.0,80% -c 400.0,40%
```

If you need to pass bang (!) characters in your command arguments, you can do so by escaping them with a backslash (). If you need to include backslashes in your command arguments, they should also be escaped with a backslash.

5.3.5 On-Demand Macros

Normally when you use host and service macros in command definitions, they refer to values for the host or service for which the command is being run. For instance, if a host check command is being executed for a host named "linuxbox", all the *standard host macros* will refer to values for that host ("linuxbox").

If you would like to reference values for another host or service in a command (for which the command is not being run), you can use what are called "on-demand" macros. On-demand macros look like normal macros, except for the fact that they contain an identifier for the host or service from which they should get their value. Here's the basic format for on-demand macros:

- "$HOSTMACRONAME:host_name$"
- "$SERVICEMACRONAME:host_name:service_description$"

Replace "HOSTMACRONAME" and "SERVICEMACRONAME" with the name of one of the standard host of service macros found *here*.

Note that the macro name is separated from the host or service identifier by a colon (:). For on-demand service macros, the service identifier consists of both a host name and a service description - these are separated by a colon (:) as well.

On-demand service macros can contain an empty host name field. In this case the name of the host associated with the service will automatically be used.

Examples of on-demand host and service macros follow:

"$HOSTDOWNTIME:myhost$ <— On-demand host macro"

"$SERVICESTATEID:novellserver:DS Database$ <— On-demand service macro"

"$SERVICESTATEID::CPU Load$ <— On-demand service macro with blank host name field"

On-demand macros are also available for hostgroup, servicegroup, contact, and contactgroup macros. For example:

"$CONTACTEMAIL:john$ <— On-demand contact macro"

"$CONTACTGROUPMEMBERS:linux-admins$ <— On-demand contactgroup macro"

"$HOSTGROUPALIAS:linux-servers$ <— On-demand hostgroup macro"

"$SERVICEGROUPALIAS:DNS-Cluster$ <— On-demand servicegroup macro"

5.3.6 On-Demand Group Macros

You can obtain the values of a macro across all contacts, hosts, or services in a specific group by using a special format for your on-demand macro declaration. You do this by referencing a specific host group, service group, or contact group name in an on-demand macro, like so:

- "$HOSTMACRONAME:hostgroup_name:delimiter$"
- "$SERVICEMACRONAME:servicegroup_name:delimiter$"
- "$CONTACTMACRONAME:contactgroup_name:delimiter$"

Replace "HOSTMACRONAME", "SERVICEMACRONAME", and "CONTACTMACRONAME" with the name of one of the standard host, service, or contact macros found *here*. The delimiter you specify is used to separate macro values for each group member.

For example, the following macro will return a comma-separated list of host state ids for hosts that are members of the hg1 hostgroup:

```
"$HOSTSTATEID:hg1:,$"
```

This macro definition will return something that looks like this:

```
"0,2,1,1,0,0,2"
```

5.3.7 Custom Variable Macros

Any *custom object variables* that you define in host, service, or contact definitions are also available as macros. Custom variable macros are named as follows:

- "$_HOSTvarname$"
- "$_SERVICEvarname$"
- "$_CONTACTvarname$"

Take the following host definition with a custom variable called "'_MACADDRESS""...

```
define host{
  host_name    linuxbox
  address    192.168.1.1
  _MACADDRESS    00:01:02:03:04:05
  ...
}
```

The "_MACADDRESS" custom variable would be available in a macro called "$_HOSTMACADDRESS$". More information on custom object variables and how they can be used in macros can be found *here*.

5.3.8 Macro Cleansing

Some macros are stripped of potentially dangerous shell metacharacters before being substituted into commands to be executed. Which characters are stripped from the macros depends on the setting of the illegal_macro_output_chars directive. The following macros are stripped of potentially dangerous characters:

- *$HOSTOUTPUT$*
- *$LONGHOSTOUTPUT$*
- *$HOSTPERFDATA$*
- *$HOSTACKAUTHOR$*
- *$HOSTACKCOMMENT$*
- *$SERVICEOUTPUT$*
- *$LONGSERVICEOUTPUT$*
- *$SERVICEPERFDATA$*
- *$SERVICEACKAUTHOR$*
- *$SERVICEACKCOMMENT$*

5.3.9 Macros as Environment Variables

Most macros are made available as environment variables for easy reference by scripts or commands that are executed by Shinken. For purposes of security and sanity, $USERn$ and "on-demand" host and service macros are not made available as environment variables.

Environment variables that contain standard macros are named the same as their corresponding macro names (listed *here*), with "**NAGIOS_**" prepended to their names. For example, the $HOSTNAME$ macro would be available as an environment variable named "NAGIOS_HOSTNAME".

5.3.10 Available Macros

A list of all the macros that are available in Shinken, as well as a chart of when they can be used, can be found *here*.

5.4 Standard Macros in Shinken

Standard macros that are available in Shinken are listed here. On-demand macros and macros for custom variables are described *here*.

5.4.1 Macro Validity

Although macros can be used in all commands you define, not all macros may be "valid" in a particular type of command. For example, some macros may only be valid during service notification commands, whereas other may only be valid during host check commands. There are ten types of commands that Nagios recognizes and treats differently. They are as follows:

- Service checks
- Service notifications
- Host checks
- Host notifications
- Service *event handlers* and/or a global service event handler
- Host *event handlers* and/or a global host event handler
- *OCSP* command
- *OCHP* command
- Service *performance data* commands
- Host *performance data* commands

The tables below list all macros currently available in Shinken, along with a brief description of each and the types of commands in which they are valid. If a macro is used in a command in which it is invalid, it is replaced with an empty string. It should be noted that macros consist of all uppercase characters and are enclosed in $ characters.

5.4.2 Macro Availability Chart

Legend:

No	The macro is not available
Yes	The macro is available

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications
Host Macros: *3*				
$HOSTNAME$	Yes	Yes	Yes	Yes

Table 5.1 – continued from previous pa

$HOSTDISPLAYNAME$	Yes	Yes	Yes	Yes
$HOSTALIAS$	Yes	Yes	Yes	Yes
$HOSTADDRESS$	Yes	Yes	Yes	Yes
$HOSTSTATE$	Yes	Yes	Yes *1*	Yes
$HOSTSTATEID$	Yes	Yes	Yes *1*	Yes
$LASTHOSTSTATE$	Yes	Yes	Yes	Yes
$LASTHOSTSTATEID$	Yes	Yes	Yes	Yes
$HOSTSTATETYPE$	Yes	Yes	Yes *1*	Yes
$HOSTATTEMPT$	Yes	Yes	Yes	Yes
$MAXHOSTATTEMPTS$	Yes	Yes	Yes	Yes
$HOSTEVENTID$	Yes	Yes	Yes	Yes
$LASTHOSTEVENTID$	Yes	Yes	Yes	Yes
$HOSTPROBLEMID$	Yes	Yes	Yes	Yes
$LASTHOSTPROBLEMID$	Yes	Yes	Yes	Yes
$HOSTLATENCY$	Yes	Yes	Yes	Yes
$HOSTEXECUTIONTIME$	Yes	Yes	Yes *1*	Yes
$HOSTDURATION$	Yes	Yes	Yes	Yes
$HOSTDURATIONSEC$	Yes	Yes	Yes	Yes
$HOSTDOWNTIME$	Yes	Yes	Yes	Yes
$HOSTPERCENTCHANGE$	Yes	Yes	Yes	Yes
$HOSTGROUPNAME$	Yes	Yes	Yes	Yes
$HOSTGROUPNAMES$	Yes	Yes	Yes	Yes
$LASTHOSTCHECK$	Yes	Yes	Yes	Yes
$LASTHOSTSTATECHANGE$	Yes	Yes	Yes	Yes
$LASTHOSTUP$	Yes	Yes	Yes	Yes
$LASTHOSTDOWN$	Yes	Yes	Yes	Yes
$LASTHOSTUNREACHABLE$	Yes	Yes	Yes	Yes
$HOSTOUTPUT$	Yes	Yes	Yes *1*	Yes
$LONGHOSTOUTPUT$	Yes	Yes	Yes *1*	Yes
$HOSTPERFDATA$	Yes	Yes	Yes *1*	Yes
$HOSTCHECKCOMMAND$	Yes	Yes	Yes	Yes
$HOSTACKAUTHOR$ 8	No	No	No	Yes
$HOSTACKAUTHORNAME$ 8	No	No	No	Yes
$HOSTACKAUTHORALIAS$ 8	No	No	No	Yes
$HOSTACKCOMMENT$ 8	No	No	No	Yes
$HOSTACTIONURL$	Yes	Yes	Yes	Yes
$HOSTNOTESURL$	Yes	Yes	Yes	Yes
$HOSTNOTES$	Yes	Yes	Yes	Yes
$TOTALHOSTSERVICES$	Yes	Yes	Yes	Yes
$TOTALHOSTSERVICESOK$	Yes	Yes	Yes	Yes
$TOTALHOSTSERVICESWARNING$	Yes	Yes	Yes	Yes
$TOTALHOSTSERVICESUNKNOWN$	Yes	Yes	Yes	Yes
$TOTALHOSTSERVICESCRITICAL$	Yes	Yes	Yes	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications
Host Group Macros:				
$HOSTGROUPALIAS$ 5	Yes	Yes	Yes	Yes
$HOSTGROUPMEMBERS$ 5	Yes	Yes	Yes	Yes
$HOSTGROUPNOTES$ 5	Yes	Yes	Yes	Yes
$HOSTGROUPNOTESURL$ 5	Yes	Yes	Yes	Yes
$HOSTGROUPACTIONURL$ 5	Yes	Yes	Yes	Yes

Table 5.1 – continued from previous pa

Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications
Service Macros:				
$SERVICEDESC$	Yes	Yes	No	No
$SERVICEDISPLAYNAME$	Yes	Yes	No	No
$SERVICESTATE$	Yes *2*	Yes	No	No
$SERVICESTATEID$	Yes *2*	Yes	No	No
$LASTSERVICESTATE$	Yes	Yes	No	No
$LASTSERVICESTATEID$	Yes	Yes	No	No
$SERVICESTATETYPE$	Yes	Yes	No	No
$SERVICEATTEMPT$	Yes	Yes	No	No
$MAXSERVICEATTEMPTS$	Yes	Yes	No	No
$SERVICEISVOLATILE$	Yes	Yes	No	No
$SERVICEEVENTID$	Yes	Yes	No	No
$LASTSERVICEEVENTID$	Yes	Yes	No	No
$SERVICEPROBLEMID$	Yes	Yes	No	No
$LASTSERVICEPROBLEMID$	Yes	Yes	No	No
$SERVICELATENCY$	Yes	Yes	No	No
$SERVICEEXECUTIONTIME$	Yes *2*	Yes	No	No
$SERVICEDURATION$	Yes	Yes	No	No
$SERVICEDURATIONSEC$	Yes	Yes	No	No
$SERVICEDOWNTIME$	Yes	Yes	No	No
$SERVICEPERCENTCHANGE$	Yes	Yes	No	No
$SERVICEGROUPNAME$	Yes	Yes	No	No
$SERVICEGROUPNAMES$	Yes	Yes	No	No
$LASTSERVICECHECK$	Yes	Yes	No	No
$LASTSERVICESTATECHANGE$	Yes	Yes	No	No
$LASTSERVICEOK$	Yes	Yes	No	No
$LASTSERVICEWARNING$	Yes	Yes	No	No
$LASTSERVICEUNKNOWN$	Yes	Yes	No	No
$LASTSERVICECRITICAL$	Yes	Yes	No	No
$SERVICEOUTPUT$	Yes *2*	Yes	No	No
$LONGSERVICEOUTPUT$	Yes *2*	Yes	No	No
$SERVICEPERFDATA$	Yes *2*	Yes	No	No
$SERVICECHECKCOMMAND$	Yes	Yes	No	No
$SERVICEACKAUTHOR$ 8	No	Yes	No	No
$SERVICEACKAUTHORNAME$ 8	No	Yes	No	No
$SERVICEACKAUTHORALIAS$ 8	No	Yes	No	No
$SERVICEACKCOMMENT$ 8	No	Yes	No	No
$SERVICEACTIONURL$	Yes	Yes	No	No
$SERVICENOTESURL$	Yes	Yes	No	No
$SERVICENOTES$	Yes	Yes	No	No
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications
Service Group Macros:				
$SERVICEGROUPALIAS$ 6	Yes	Yes	Yes	Yes
$SERVICEGROUPMEMBERS$ 6	Yes	Yes	Yes	Yes
$SERVICEGROUPNOTES$ 6	Yes	Yes	Yes	Yes
$SERVICEGROUPNOTESURL$ 6	Yes	Yes	Yes	Yes
$SERVICEGROUPACTIONURL$ 6	Yes	Yes	Yes	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications
Contact Macros:				

Table 5.1 – continued from previous pa

$CONTACTNAME$	No	Yes	No	Yes
$CONTACTALIAS$	No	Yes	No	Yes
$CONTACTEMAIL$	No	Yes	No	Yes
$CONTACTPAGER$	No	Yes	No	Yes
$CONTACTADDRESSn$	No	Yes	No	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications
Contact Group Macros:				
$CONTACTGROUPALIAS$ 7	Yes	Yes	Yes	Yes
$CONTACTGROUPMEMBERS$ 7	Yes	Yes	Yes	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications
Summary Macros:				
$TOTALHOSTSUP$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALHOSTSDOWN$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALHOSTSUNREACHABLE$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALHOSTSDOWNUNHANDLED$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALHOSTSUNREACHABLEUNHANDLED$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALHOSTPROBLEMS$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALHOSTPROBLEMSUNHANDLED$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALSERVICESOK$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALSERVICESWARNING$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALSERVICESCRITICAL$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALSERVICESUNKNOWN$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALSERVICESWARNINGUNHANDLED$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALSERVICESCRITICALUNHANDLED$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALSERVICESUNKNOWNUNHANDLED$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALSERVICEPROBLEMS$ 10	Yes	Yes *4*	Yes	Yes *4*
$TOTALSERVICEPROBLEMSUNHANDLED$ 10	Yes	Yes *4*	Yes	Yes *4*
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications
Notification Macros:				
$NOTIFICATIONTYPE$	No	Yes	No	Yes
$NOTIFICATIONRECIPIENTS$	No	Yes	No	Yes
$NOTIFICATIONISESCALATED$	No	Yes	No	Yes
$NOTIFICATIONAUTHOR$	No	Yes	No	Yes
$NOTIFICATIONAUTHORNAME$	No	Yes	No	Yes
$NOTIFICATIONAUTHORALIAS$	No	Yes	No	Yes
$NOTIFICATIONCOMMENT$	No	Yes	No	Yes
$HOSTNOTIFICATIONNUMBER$	No	Yes	No	Yes
$HOSTNOTIFICATIONID$	No	Yes	No	Yes
$SERVICENOTIFICATIONNUMBER$	No	Yes	No	Yes
$SERVICENOTIFICATIONID$	No	Yes	No	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications
Date/Time Macros:				
$LONGDATETIME$	Yes	Yes	Yes	Yes
$SHORTDATETIME$	Yes	Yes	Yes	Yes
$DATE$	Yes	Yes	Yes	Yes
$TIME$	Yes	Yes	Yes	Yes
$TIMET$	Yes	Yes	Yes	Yes
$ISVALIDTIME:$	Yes	Yes	Yes	Yes
$NEXTVALIDTIME:$	Yes	Yes	Yes	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications

Table 5.1 – continued from previous pa

File Macros:				
$MAINCONFIGFILE$	Yes	Yes	Yes	Yes
$STATUSDATAFILE$	Yes	Yes	Yes	Yes
$COMMENTDATAFILE$	Yes	Yes	Yes	Yes
$DOWNTIMEDATAFILE$	Yes	Yes	Yes	Yes
$RETENTIONDATAFILE$	Yes	Yes	Yes	Yes
$OBJECTCACHEFILE$	Yes	Yes	Yes	Yes
$TEMPFILE$	Yes	Yes	Yes	Yes
$TEMPPATH$	Yes	Yes	Yes	Yes
$LOGFILE$	Yes	Yes	Yes	Yes
$RESOURCEFILE$	Yes	Yes	Yes	Yes
$COMMANDFILE$	Yes	Yes	Yes	Yes
$HOSTPERFDATAFILE$	Yes	Yes	Yes	Yes
$SERVICEPERFDATAFILE$	Yes	Yes	Yes	Yes
Macro Name	Service Checks	Service Notifications	Host Checks	Host Notifications
Misc Macros:				
$PROCESSSTARTTIME$	Yes	Yes	Yes	Yes
$EVENTSTARTTIME$	Yes	Yes	Yes	Yes
$ADMINEMAIL$	Yes	Yes	Yes	Yes
$ADMINPAGER$	Yes	Yes	Yes	Yes
$ARGn$	Yes	Yes	Yes	Yes
$USERn$	Yes	Yes	Yes	Yes

5.4.3 Macro Descriptions

Host Macros: *3*	
$HOSTNAME$	Short name for the host (i.e. "biglinuxbox"). This value is taken from the host_
$HOSTDISPLAYNAME$	An alternate display name for the host. This value is taken from the display_na
$HOSTALIAS$	Long name/description for the host. This value is taken from the alias directive
$HOSTADDRESS$	Address of the host. This value is taken from the address directive in the *host d*
$HOSTSTATE$	A string indicating the current state of the host ("UP", "DOWN", or "UNREA
$HOSTSTATEID$	A number that corresponds to the current state of the host: 0=UP, 1=DOWN, 2
$LASTHOSTSTATE$	A string indicating the last state of the host ("UP", "DOWN", or "UNREACH
$LASTHOSTSTATEID$	A number that corresponds to the last state of the host: 0=UP, 1=DOWN, 2=U
$HOSTSTATETYPE$	A string indicating the *state type* for the current host check ("HARD" or "SOF
$HOSTATTEMPT$	The number of the current host check retry. For instance, if this is the second t
$MAXHOSTATTEMPTS$	The max check attempts as defined for the current host. Useful when writing h
$HOSTEVENTID$	A globally unique number associated with the host's current state. Every time
$LASTHOSTEVENTID$	The previous (globally unique) event number that was given to the host.
$HOSTPROBLEMID$	A globally unique number associated with the host's current problem state. Ev
$LASTHOSTPROBLEMID$	The previous (globally unique) problem number that was given to the host. Co
$HOSTLATENCY$	A (floating point) number indicating the number of seconds that a scheduled h
$HOSTEXECUTIONTIME$	A (floating point) number indicating the number of seconds that the host check
$HOSTDURATION$	A string indicating the amount of time that the host has spent in its current stat
$HOSTDURATIONSEC$	A number indicating the number of seconds that the host has spent in its curre
$HOSTDOWNTIME$	A number indicating the current "downtime depth" for the host. If this host is
$HOSTPERCENTCHANGE$	A (floating point) number indicating the percent state change the host has unde
$HOSTGROUPNAME$	The short name of the hostgroup that this host belongs to. This value is taken f
$HOSTGROUPNAMES$	A comma separated list of the short names of all the hostgroups that this host b

$LASTHOSTCHECK$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicatir
$LASTHOSTSTATECHANGE$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicatir
$LASTHOSTUP$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicatir
$LASTHOSTDOWN$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicatir
$LASTHOSTUNREACHABLE$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicatir
$HOSTOUTPUT$	The first line of text output from the last host check (i.e. "Ping OK").
$LONGHOSTOUTPUT$	The full text output (aside from the first line) from the last host check.
$HOSTPERFDATA$	This macro contains any *performance data* that may have been returned by the
$HOSTCHECKCOMMAND$	This macro contains the name of the command (along with any arguments pas
$HOSTACKAUTHOR$ *8*	A string containing the name of the user who acknowledged the host problem.
$HOSTACKAUTHORNAME$ *8*	A string containing the short name of the contact (if applicable) who acknowle
$HOSTACKAUTHORALIAS$ *8*	A string containing the alias of the contact (if applicable) who acknowledged t
$HOSTACKCOMMENT$ *8*	A string containing the acknowledgement comment that was entered by the us
$HOSTACTIONURL$	Action URL for the host. This macro may contain other macros (e.g. $HOSTN
$HOSTNOTESURL$	Notes URL for the host. This macro may contain other macros (e.g. $HOSTN
$HOSTNOTES$	Notes for the host. This macro may contain other macros (e.g. $HOSTNAME
$TOTALHOSTSERVICES$	The total number of services associated with the host.
$TOTALHOSTSERVICESOK$	The total number of services associated with the host that are in an OK state.
$TOTALHOSTSERVICESWARNING$	The total number of services associated with the host that are in a WARNING
$TOTALHOSTSERVICESUNKNOWN$	The total number of services associated with the host that are in an UNKNOW
$TOTALHOSTSERVICESCRITICAL$	The total number of services associated with the host that are in a CRITICAL
Host Group Macros: *5*	
$HOSTGROUPALIAS$ *5*	The long name / alias of either 1) the hostgroup name passed as an on-demand
$HOSTGROUPMEMBERS$ *5*	A comma-separated list of all hosts that belong to either 1) the hostgroup name
$HOSTGROUPNOTES$ *5*	The notes associated with either 1) the hostgroup name passed as an on-deman
$HOSTGROUPNOTESURL$ *5*	The notes URL associated with either 1) the hostgroup name passed as an on-d
$HOSTGROUPACTIONURL$ *5*	The action URL associated with either 1) the hostgroup name passed as an on-
Service Macros:	
$SERVICEDESC$	The long name/description of the service (i.e. "Main Website"). This value is t
$SERVICEDISPLAYNAME$	An alternate display name for the service. This value is taken from the display
$SERVICESTATE$	A string indicating the current state of the service ("OK", "WARNING", "UNI
$SERVICESTATEID$	A number that corresponds to the current state of the service: 0=OK, 1=WARN
$LASTSERVICESTATE$	A string indicating the last state of the service ("OK", "WARNING", "UNKN
$LASTSERVICESTATEID$	A number that corresponds to the last state of the service: 0=OK, 1=WARNIN
$SERVICESTATETYPE$	A string indicating the *state type* for the current service check ("HARD" or "S
$SERVICEATTEMPT$	The number of the current service check retry. For instance, if this is the secon
$MAXSERVICEATTEMPTS$	The max check attempts as defined for the current service. Useful when writin
$SERVICEISVOLATILE$	Indicates whether the service is marked as being volatile or not: 0 = not volatil
$SERVICEEVENTID$	A globally unique number associated with the service's current state. Every tir
$LASTSERVICEEVENTID$	The previous (globally unique) event number that given to the service.
$SERVICEPROBLEMID$	A globally unique number associated with the service's current problem state.
$LASTSERVICEPROBLEMID$	The previous (globally unique) problem number that was given to the service.
$SERVICELATENCY$	A (floating point) number indicating the number of seconds that a scheduled se
$SERVICEEXECUTIONTIME$	A (floating point) number indicating the number of seconds that the service ch
$SERVICEDURATION$	A string indicating the amount of time that the service has spent in its current s
$SERVICEDURATIONSEC$	A number indicating the number of seconds that the service has spent in its cur
$SERVICEDOWNTIME$	A number indicating the current "downtime depth" for the service. If this servi
$SERVICEPERCENTCHANGE$	A (floating point) number indicating the percent state change the service has u
$SERVICEGROUPNAME$	The short name of the servicegroup that this service belongs to. This value is t
$SERVICEGROUPNAMES$	A comma separated list of the short names of all the servicegroups that this ser

$LASTSERVICECHECK$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicatir
$LASTSERVICESTATECHANGE$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicatir
$LASTSERVICEOK$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicatir
$LASTSERVICEWARNING$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicatir
$LASTSERVICEUNKNOWN$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicatir
$LASTSERVICECRITICAL$	This is a timestamp in time_t format (seconds since the UNIX epoch) indicatir
$SERVICEOUTPUT$	The first line of text output from the last service check (i.e. "Ping OK").
$LONGSERVICEOUTPUT$	The full text output (aside from the first line) from the last service check.
$SERVICEPERFDATA$	This macro contains any *performance data* that may have been returned by the
$SERVICECHECKCOMMAND$	This macro contains the name of the command (along with any arguments pas
$SERVICEACKAUTHOR$ *8*	A string containing the name of the user who acknowledged the service proble
$SERVICEACKAUTHORNAME$ *8*	A string containing the short name of the contact (if applicable) who acknowle
$SERVICEACKAUTHORALIAS$ *8*	A string containing the alias of the contact (if applicable) who acknowledged t
$SERVICEACKCOMMENT$ *8*	A string containing the acknowledgement comment that was entered by the us
$SERVICEACTIONURL$	Action URL for the service. This macro may contain other macros (e.g. $HOS
$SERVICENOTESURL$	Notes URL for the service. This macro may contain other macros (e.g. $HOST
$SERVICENOTES$	Notes for the service. This macro may contain other macros (e.g. $HOSTNAM
Service Group Macros: *6*	
$SERVICEGROUPALIAS$ *6*	The long name / alias of either 1) the servicegroup name passed as an on-dema
$SERVICEGROUPMEMBERS$ *6*	A comma-separated list of all services that belong to either 1) the servicegroup
$SERVICEGROUPNOTES$ *6*	The notes associated with either 1) the servicegroup name passed as an on-den
$SERVICEGROUPNOTESURL$ *6*	The notes URL associated with either 1) the servicegroup name passed as an o
$SERVICEGROUPACTIONURL$ *6*	The action URL associated with either 1) the servicegroup name passed as an
Contact Macros:	
$CONTACTNAME$	Short name for the contact (i.e. "jdoe") that is being notified of a host or servic
$CONTACTALIAS$	Long name/description for the contact (i.e. "John Doe") being notified. This va
$CONTACTEMAIL$	Email address of the contact being notified. This value is taken from the email
$CONTACTPAGER$	Pager number/address of the contact being notified. This value is taken from th
$CONTACTADDRESSn$	Address of the contact being notified. Each contact can have six different addr
$CONTACTGROUPNAME$	The short name of the contactgroup that this contact is a member of. This valu
$CONTACTGROUPNAMES$	A comma separated list of the short names of all the contactgroups that this co
Contact Group Macros: *5*	
$CONTACTGROUPALIAS$ *7*	The long name / alias of either 1) the contactgroup name passed as an on-dema
$CONTACTGROUPMEMBERS$ *7*	A comma-separated list of all contacts that belong to either 1) the contactgroup
SUMMARY Macros:	
$TOTALHOSTSUP$	This macro reflects the total number of hosts that are currently in an UP state.
$TOTALHOSTSDOWN$	This macro reflects the total number of hosts that are currently in a DOWN sta
$TOTALHOSTSUNREACHABLE$	This macro reflects the total number of hosts that are currently in an UNREAC
$TOTALHOSTSDOWNUNHANDLED$	This macro reflects the total number of hosts that are currently in a DOWN sta
$TOTALHOSTSUNREACHABLEUNHANDLED$	This macro reflects the total number of hosts that are currently in an UNREAC
$TOTALHOSTPROBLEMS$	This macro reflects the total number of hosts that are currently either in a DOW
$TOTALHOSTPROBLEMSUNHANDLED$	This macro reflects the total number of hosts that are currently either in a DOW
$TOTALSERVICESOK$	This macro reflects the total number of services that are currently in an OK sta
$TOTALSERVICESWARNING$	This macro reflects the total number of services that are currently in a WARNI
$TOTALSERVICESCRITICAL$	This macro reflects the total number of services that are currently in a CRITIC
$TOTALSERVICESUNKNOWN$	This macro reflects the total number of services that are currently in an UNKN
$TOTALSERVICESWARNINGUNHANDLED$	This macro reflects the total number of services that are currently in a WARNI
$TOTALSERVICESCRITICALUNHANDLED$	This macro reflects the total number of services that are currently in a CRITIC
$TOTALSERVICESUNKNOWNUNHANDLED$	This macro reflects the total number of services that are currently in an UNKN
$TOTALSERVICEPROBLEMS$	This macro reflects the total number of services that are currently either in a W

$TOTALSERVICEPROBLEMSUNHANDLED$	This macro reflects the total number of services that are currently either in a W
Notification Macros:	
$NOTIFICATIONTYPE$	A string identifying the type of notification that is being sent ("PROBLEM", "
$NOTIFICATIONRECIPIENTS$	A comma-separated list of the short names of all contacts that are being notifie
$NOTIFICATIONISESCALATED$	An integer indicating whether this was sent to normal contacts for the host or s
$NOTIFICATIONAUTHOR$	A string containing the name of the user who authored the notification. If the $
$NOTIFICATIONAUTHORNAME$	A string containing the short name of the contact (if applicable) specified in th
$NOTIFICATIONAUTHORALIAS$	A string containing the alias of the contact (if applicable) specified in the $NO
$NOTIFICATIONCOMMENT$	A string containing the comment that was entered by the notification author. If
$HOSTNOTIFICATIONNUMBER$	The current notification number for the host. The notification number increase
$HOSTNOTIFICATIONID$	A unique number identifying a host notification. Notification ID numbers are u
$SERVICENOTIFICATIONNUMBER$	The current notification number for the service. The notification number increa
$SERVICENOTIFICATIONID$	A unique number identifying a service notification. Notification ID numbers ar
Date/Time Macros:	
$LONGDATETIME$	Current date/time stamp (i.e. Fri Oct 13 00:30:28 CDT 2000). Format of date i
$SHORTDATETIME$	Current date/time stamp (i.e. 10-13-2000 00:30:28). Format of date is determi
$DATE$	Date stamp (i.e. 10-13-2000). Format of date is determined by *date_format* di
$TIME$	Current time stamp (i.e. 00:30:28).
$TIMET$	Current time stamp in time_t format (seconds since the UNIX epoch).
$ISVALIDTIME:$ *9*	This is a special on-demand macro that returns a 1 or 0 depending on whether
$NEXTVALIDTIME:$ *9*	This is a special on-demand macro that returns the next valid time (in time_t fo
File Macros:	
$MAINCONFIGFILE$	The location of the *main config file*.
$STATUSDATAFILE$	The location of the status data file.
$COMMENTDATAFILE$	The location of the comment data file.
$DOWNTIMEDATAFILE$	The location of the downtime data file.
$RETENTIONDATAFILE$	The location of the retention data file.
$OBJECTCACHEFILE$	The location of the object cache file.
$TEMPFILE$	The location of the temp file.
$TEMPPATH$	The directory specified by the temp path variable.
$LOGFILE$	The location of the *log file*.
$RESOURCEFILE$	The location of the *resource file*.
$COMMANDFILE$	The location of the *command file*.
$HOSTPERFDATAFILE$	The location of the host performance data file (if defined).
$SERVICEPERFDATAFILE$	The location of the service performance data file (if defined).
Misc Macros:	
$PROCESSSTARTTIME$	Time stamp in time_t format (seconds since the UNIX epoch) indicating when
$EVENTSTARTTIME$	Time stamp in time_t format (seconds since the UNIX epoch) indicating when
$ADMINEMAIL$	Global administrative email address. This value is taken from the admin_emai
$ADMINPAGER$	Global administrative pager number/address. This value is taken from the adm
$ARGn$	The nth argument passed to the command (notification, event handler, service
$USERn$	The nth user-definable macro. User macros can be defined in one or more *reso*

5.4.4 Notes

-1 These macros are not valid for the host they are associated with when that host is being checked (i.e. they make no sense, as they haven't been determined yet). -2 These macros are not valid for the service they are associated with when that service is being checked (i.e. they make no sense, as they haven't been determined yet). -3 When host macros are used in service-related commands (i.e. service notifications, event handlers, etc) they refer to the host that the service is associated with. -4 When host and service summary macros are used in notification commands, the totals are filtered to reflect only those hosts and services for which the contact is authorized (i.e. hosts and services

they are configured to receive notifications for). -5 These macros are normally associated with the first/primary hostgroup associated with the current host. They could therefore be considered host macros in many cases. However, these macros are not available as on-demand host macros. Instead, they can be used as on-demand hostgroup macros when you pass the name of a hostgroup to the macro. For example: $HOSTGROUPMEMBERS:hg1$ would return a comma-delimited list of all (host) members of the hostgroup hg1. -6 These macros are normally associated with the first/primary servicegroup associated with the current service. They could therefore be considered service macros in many cases. However, these macros are not available as on-demand service macros. Instead, they can be used as on-demand servicegroup macros when you pass the name of a servicegroup to the macro. For example: $SERVICEGROUPMEMBERS:sg1$ would return a comma-delimited list of all (service) members of the servicegroup sg1. -7 These macros are normally associated with the first/primary contactgroup associated with the current contact. They could therefore be considered contact macros in many cases. However, these macros are not available as on-demand contact macros. Instead, they can be used as on-demand contactgroup macros when you pass the name of a contactgroup to the macro. For example: $CONTACTGROUPMEMBERS:cg1$ would return a comma-delimited list of all (contact) members of the contactgroup cg1. -8 These acknowledgement macros are deprecated. Use the more generic $NOTIFICATIONAUTHOR$, $NOTIFICATIONAUTHORNAME$, $NOTIFICATIONAUTHORALIAS$ or $NOTIFICATIONAUTHORCOMMENT$ macros instead. -9 These macro are only available as on-demand macros - e.g. you must supply an additional argument with them in order to use them. These macros are not available as environment variables. -10 Summary macros are not available as environment variables if the *use_large_installation_tweaks* option is enabled, as they are quite CPU-intensive to calculate.

5.5 Host Checks

5.5.1 Introduction

The basic workings of host checks are described here...

5.5.2 When Are Host Checks Performed?

Hosts are checked by the Shinken daemon:

- At regular intervals, as defined by the check_interval and retry_interval options in your *host definitions*.
- On-demand when a service associated with the host changes state.
- On-demand as needed as part of the *host reachability* logic.
- On-demand as needed for *predictive host dependency checks*.

Regularly scheduled host checks are optional. If you set the check_interval option in your host definition to zero (0), Shinken will not perform checks of the hosts on a regular basis. It will, however, still perform on-demand checks of the host as needed for other parts of the monitoring logic.

On-demand checks are made when a service associated with the host changes state because Shinken needs to know whether the host has also changed state. Services that change state are often an indicator that the host may have also changed state. For example, if Shinken detects that the "HTTP" service associated with a host just changed from a CRITICAL to an OK state, it may indicate that the host just recovered from a reboot and is now back up and running.

On-demand checks of hosts are also made as part of the *host reachability* logic. Shinken is designed to detect network outages as quickly as possible, and distinguish between DOWN and UNREACHABLE host states. These are very different states and can help an admin quickly locate the cause of a network outage.

On-demand checks are also performed as part of the *predictive host dependency check* logic. These checks help ensure that the dependency logic is as accurate as possible.

5.5.3 Cached Host Checks

The performance of on-demand host checks can be significantly improved by implementing the use of cached checks, which allow Shinken to forgo executing a host check if it determines a relatively recent check result will do instead. More information on cached checks can be found *here*.

5.5.4 Dependencies and Checks

You can define *host execution dependencies* that prevent Shinken from checking the status of a host depending on the state of one or more other hosts. More information on dependencies can be found *here*.

5.5.5 Parallelization of Host Checks

All checks are run in parallel.

5.5.6 Host States

Hosts that are checked can be in one of three different states:

- UP
- DOWN
- UNREACHABLE

5.5.7 Host State Determination

Host checks are performed by *plugins*, which can return a state of OK, WARNING, UNKNOWN, or CRITICAL. How does Shinken translate these plugin return codes into host states of UP, DOWN, or UNREACHABLE? Lets see...

The table below shows how plugin return codes correspond with preliminary host states. Some post-processing (which is described later) is done which may then alter the final host state.

Plugin Result	Preliminary Host State
OK	UP
WARNING	DOWN*
UNKNOWN	DOWN
CRITICAL	DOWN

If the preliminary host state is DOWN, Shinken will attempt to see if the host is really DOWN or if it is UNREACHABLE. The distinction between DOWN and UNREACHABLE host states is important, as it allows admins to determine root cause of network outages faster. The following table shows how Shinken makes a final state determination based on the state of the hosts parent(s). A host's parents are defined in the parents directive in host definition.

Preliminary Host State	Parent Host State	Final Host State
DOWN	At least one parent is UP	DOWN
DOWN	All parents are either DOWN or UNREACHABLE	UNREACHABLE

More information on how Shinken distinguishes between DOWN and UNREACHABLE states can be found *here*.

5.5.8 Host State Changes

As you are probably well aware, hosts don't always stay in one state. Things break, patches get applied, and servers need to be rebooted. When Shinken checks the status of hosts, it will be able to detect when a host changes between UP, DOWN, and UNREACHABLE states and take appropriate action. These state changes result in different *state types* (HARD or SOFT), which can trigger *event handlers* to be run and *notifications* to be sent out. Detecting and dealing with state changes is what Shinken is all about.

When hosts change state too frequently they are considered to be "flapping". A good example of a flapping host would be server that keeps spontaneously rebooting as soon as the operating system loads. That's always a fun scenario to have to deal with. Shinken can detect when hosts start flapping, and can suppress notifications until flapping stops and the host's state stabilizes. More information on the flap detection logic can be found *here*.

5.6 Service Checks

5.6.1 Introduction

The basic workings of service checks are described here...

5.6.2 When Are Service Checks Performed?

Services are checked by the Shinken daemon:

- At regular intervals, as defined by the "check_interval" and "retry_interval" options in your *service definitions*.
- On-demand as needed for *predictive service dependency checks*.

On-demand checks are performed as part of the *predictive service dependency check* logic. These checks help ensure that the dependency logic is as accurate as possible. If you don't make use of *service dependencies*, Shinken won't perform any on-demand service checks.

5.6.3 Cached Service Checks

The performance of on-demand service checks can be significantly improved by implementing the use of cached checks, which allow Shinken to forgo executing a service check if it determines a relatively recent check result will do instead. Cached checks will only provide a performance increase if you are making use of *service dependencies*. More information on cached checks can be found *here*.

5.6.4 Dependencies and Checks

You can define *service execution dependencies* that prevent Shinken from checking the status of a service depending on the state of one or more other services. More information on dependencies can be found *here*.

5.6.5 Parallelization of Service Checks

Scheduled service checks are run in parallel.

5.6.6 Service States

Services that are checked can be in one of four different states:

- OK
- WARNING
- UNKNOWN
- CRITICAL

5.6.7 Service State Determination

Service checks are performed by *plugins*, which can return a state of OK, WARNING, UNKNOWN, or CRITICAL. These plugin states directly translate to service states. For example, a plugin which returns a WARNING state will cause a service to have a WARNING state.

5.6.8 Services State Changes

When Shinken checks the status of services, it will be able to detect when a service changes between OK, WARNING, UNKNOWN, and CRITICAL states and take appropriate action. These state changes result in different *state types* (HARD or SOFT), which can trigger *event handlers* to be run and *notifications* to be sent out. Service state changes can also trigger on-demand *host checks*. Detecting and dealing with state changes is what Shinken is all about.

When services change state too frequently they are considered to be "flapping". Shinken can detect when services start flapping, and can suppress notifications until flapping stops and the service's state stabilizes. More information on the flap detection logic can be found *here*.

5.7 Active Checks

5.7.1 Introduction

Shinken is capable of monitoring hosts and services in two ways: actively and passively. Passive checks are described *elsewhere*, so we'll focus on active checks here. Active checks are the most common method for monitoring hosts and services. The main features of actives checks are as follows:

- Active checks are initiated by the Shinken process
- Active checks are run on a regularly scheduled basis

5.7.2 How Are Active Checks Performed?

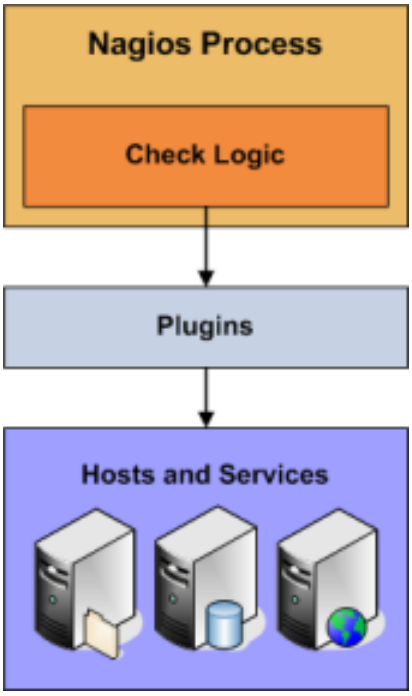

Active checks are initiated by the check logic in the Shinken daemon. When Shinken needs to check the status of a host or service it will execute a plugin and pass it information about what needs to be checked. The plugin will then check the operational state of the host or service and report the results back to the Shinken daemon. Shinken will process the results of the host or service check and take appropriate action as necessary (e.g. send notifications, run event handlers, etc).

More information on how plugins work can be found *here*.

5.7.3 When Are Active Checks Executed?

Active check are executed:

- At regular intervals, as defined by the "check_interval" and "retry_interval" options in your host and service definitions
- On-demand as needed

Regularly scheduled checks occur at intervals equaling either the "check_interval" or the "retry_interval" in your host or service definitions, depending on what *type of state* the host or service is in. If a host or service is in a HARD state, it will be actively checked at intervals equal to the "check_interval" option. If it is in a SOFT state, it will be checked at intervals equal to the retry_interval option.

On-demand checks are performed whenever Shinken sees a need to obtain the latest status information about a particular host or service. For example, when Shinken is determining the *reachability* of a host, it will often perform on-demand checks of parent and child hosts to accurately determine the status of a particular network segment. On-demand checks also occur in the *predictive dependency check* logic in order to ensure Shinken has the most accurate status information.

5.8 Passive Checks

5.8.1 Introduction

In most cases you'll use Shinken to monitor your hosts and services using regularly scheduled *active checks*. Active checks can be used to "poll" a device or service for status information every so often. Shinken also supports a way to monitor hosts and services passively instead of actively. They key features of passive checks are as follows:

- Passive checks are initiated and performed by external applications/processes
- Passive check results are submitted to Shinken for processing

The major difference between active and passive checks is that active checks are initiated and performed by Shinken, while passive checks are performed by external applications.

5.8.2 Uses For Passive Checks

Passive checks are useful for monitoring services that are:

- Asynchronous in nature, they cannot or would not be monitored effectively by polling their status on a regularly scheduled basis
- Located behind a firewall and cannot be checked actively from the monitoring host

Examples of asynchronous services that lend themselves to being monitored passively include:

- "SNMP" traps and security alerts. You never know how many (if any) traps or alerts you'll receive in a given time frame, so it's not feasible to just monitor their status every few minutes.
- Aggregated checks from a host running an agent. Checks may be run at much lower intervals on hosts running an agent.
- Submitting check results that happen directly within an application without using an intermediate log file(syslog, event log, etc.).

Passive checks are also used when configuring *distributed* or *redundant* monitoring installations.

5.8.3 How Passive Checks Work

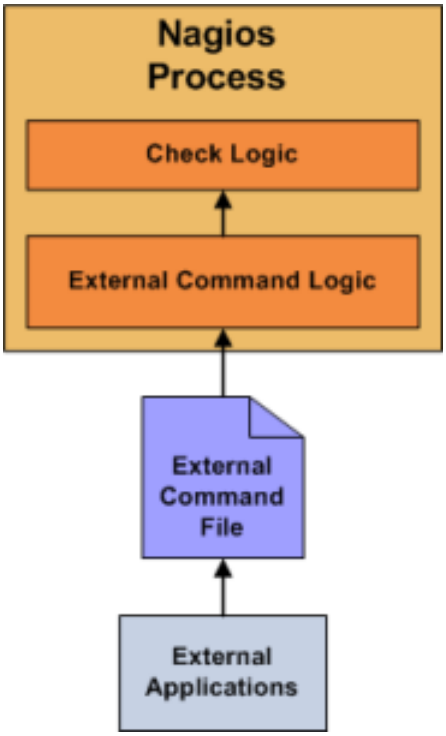

DEPRECATED IMAGE - TODO REPLACE WITH MOE ACCURATE DEPTICTION

Here's how passive checks work in more detail...

- An external application checks the status of a host or service.
- The external application writes the results of the check to the *external command named pipe* (a named pipe is a "memory pipe", so there is no disk IO involved).
- Shinken reads the external command file and places the results of all passive checks into a queue for processing by the appropriate process in the Shinken cloud.
- Shinken will execute a check result reaper event each second and scan the check result queue. Each service check result that is found in the queue is processed in the same manner - regardless of whether the check was active or passive. Shinken may send out notifications, log alerts, etc. depending on the check result information.

The processing of active and passive check results is essentially identical. This allows for seamless integration of status information from external applications with Shinken.

5.8.4 Enabling Passive Checks

In order to enable passive checks in Shinken, you'll need to do the following:

- Set *"accept_passive_service_checks"* directive is set to 1 (in nagios.cfg).
- Set the "passive_checks_enabled" directive in your host and service definitions is set to 1.

If you want to disable processing of passive checks on a global basis, set the *"accept_passive_service_checks"* directive to 0.

If you would like to disable passive checks for just a few hosts or services, use the "passive_checks_enabled" directive in the host and/or service definitions to do so.

5.8.5 Submitting Passive Service Check Results

External applications can submit passive service check results to Shinken by writing a PROCESS_SERVICE_CHECK_RESULT *external command* to the external command pipe, which is essentially a file handle that you write to as you would a file.

The format of the command is as follows: "[<timestamp>] PROCESS_SERVICE_CHECK_RESULT;<host_name>;<svc_description>;<
where...

- timestamp is the time in time_t format (seconds since the UNIX epoch) that the service check was perfomed (or submitted). Please note the single space after the right bracket.
- host_name is the short name of the host associated with the service in the service definition
- svc_description is the description of the service as specified in the service definition
- return_code is the return code of the check (0=OK, 1=WARNING, 2=CRITICAL, 3=UNKNOWN)
- plugin_output is the text output of the service check (i.e. the plugin output)

A service must be defined in Shinken before Shinken will accept passive check results for it! Shinken will ignore all check results for services that have not been configured before it was last (re)started.

An example shell script of how to submit passive service check results to Shinken can be found in the documentation on *volatile services*.

5.8.6 Submitting Passive Host Check Results

External applications can submit passive host check results to Shinken by writing a PROCESS_HOST_CHECK_RESULT external command to the external command file.

The format of the command is as follows: "[<timestamp>]PROCESS_HOST_CHECK_RESULT;<host_name>;<host_status>;<plugin_o
where...

- timestamp is the time in time_t format (seconds since the UNIX epoch) that the host check was perfomed (or submitted). Please note the single space after the right bracket.
- host_name is the short name of the host (as defined in the host definition)
- host_status is the status of the host (0=UP, 1=DOWN, 2=UNREACHABLE)
- plugin_output is the text output of the host check

A host must be defined in Shinken before you can submit passive check results for it! Shinken will ignore all check results for hosts that had not been configured before it was last (re)started.

Once data has been received by the Arbiter process, either directly or through a Receiver daemon, it will forward the check results to the appropriate Scheduler to apply check logic.

5.8.7 Passive Checks and Host States

Unlike with active host checks, Shinken does not (by default) attempt to determine whether or host is DOWN or UNREACHABLE with passive checks. Rather, Shinken takes the passive check result to be the actual state the host is in and doesn't try to determine the hosts' actual state using the *reachability logic*. This can cause problems if you are submitting passive checks from a remote host or you have a *distributed monitoring setup* where the parent/child host relationships are different.

You can tell Shinken to translate DOWN/UNREACHABLE passive check result states to their "proper" state by using the "translate_passive_host_checks" variable. More information on how this works can be found *here*.

Passive host checks are normally treated as *HARD states*, unless the "passive_host_checks_are_soft" option is enabled.

5.8.8 Submitting Passive Check Results From Remote Hosts

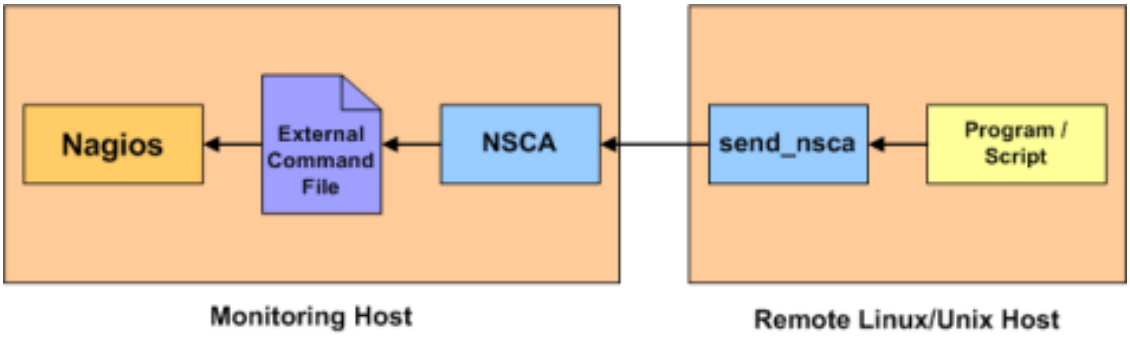

DEPRECATED IMAGE - TODO REPLACE WITH MOE ACCURATE DEPTICTION

If an application that resides on the same host as Shinken is sending passive host or service check results, it can simply write the results directly to the external command named pipe file as outlined above. However, applications on remote hosts can't do this so easily.

In order to allow remote hosts to send passive check results to the monitoring host, there a multiple modules to that can send and accept passive check results. NSCA, TSCA, Shinken WebService and more.

Learn more about the different passive check result/command protocols and how to configure them.

5.9 State Types

5.9.1 Introduction

The current state of monitored services and hosts is determined by two components:

- The status of the service or host (i.e. OK, WARNING, UP, DOWN, etc.)
- The type of state the service or host is in.

There are two state types in Shinken - SOFT states and HARD states. These state types are a crucial part of the monitoring logic, as they are used to determine when *event handlers* are executed and when *notifications* are initially sent out.

This document describes the difference between SOFT and HARD states, how they occur, and what happens when they occur.

5.9.2 Service and Host Check Retries

In order to prevent false alarms from transient problems, Shinken allows you to define how many times a service or host should be (re)checked before it is considered to have a "real" problem. This is controlled by the max_check_attempts option in the host and service definitions. Understanding how hosts and services are (re)checked in order to determine if a real problem exists is important in understanding how state types work.

5.9.3 Soft States

Soft states occur in the following situations...

- When a service or host check results in a non-OK or non-UP state and the service check has not yet been (re)checked the number of times specified by the max_check_attempts directive in the service or host definition. This is called a soft error.
- When a service or host recovers from a soft error. This is considered a soft recovery.

The following things occur when hosts or services experience SOFT state changes:

- The SOFT state is logged.
- Event handlers are executed to handle the SOFT state.

SOFT states are only logged if you enabled the log_service_retries or log_host_retries options in your main configuration file.

The only important thing that really happens during a soft state is the execution of event handlers. Using event handlers can be particularly useful if you want to try and proactively fix a problem before it turns into a HARD state. The *$HOSTSTATETYPE$* or *$SERVICESTATETYPE$* macros will have a value of "SOFT" when event handlers are executed, which allows your event handler scripts to know when they should take corrective action. More information on event handlers can be found *here*.

5.9.4 Hard States

Hard states occur for hosts and services in the following situations:

- When a host or service check results in a non-UP or non-OK state and it has been (re)checked the number of times specified by the max_check_attempts option in the host or service definition. This is a hard error state.
- When a host or service transitions from one hard error state to another error state (e.g. WARNING to CRITICAL).
- When a service check results in a non-OK state and its corresponding host is either DOWN or UNREACHABLE.
- When a host or service recovers from a hard error state. This is considered to be a hard recovery.
- When a *passive host check* is received. Passive host checks are treated as HARD unless the passive_host_checks_are_soft option is enabled.

The following things occur when hosts or services experience HARD state changes:

- The HARD state is logged.
- Event handlers are executed to handle the HARD state.
- Contacts are notifified of the host or service problem or recovery.

The *$HOSTSTATETYPE$* or *$SERVICESTATETYPE$* macros will have a value of "HARD" when event handlers are executed, which allows your event handler scripts to know when they should take corrective action. More information on event handlers can be found *here*.

5.9.5 Example

Here's an example of how state types are determined, when state changes occur, and when event handlers and notifications are sent out. The table below shows consecutive checks of a service over time. The service has a max_check_attempts value of 3.

Time	Check #	State	State Type	State Change	Notes
0	1	OK	HARD	No	Initial state of the service
1	1	CRITICAL	SOFT	Yes	First detection of a non-OK state. Event handlers execute.
2	2	WARNING	SOFT	Yes	Service continues to be in a non-OK state. Event handlers execute.
3	3	CRITICAL	HARD	Yes	Max check attempts has been reached, so service goes into a HARD state. Event handlers execute and a problem notification is sent out. Check # is reset to 1 immediately after this happens.
4	1	WARNING	HARD	Yes	Service changes to a HARD WARNING state. Event handlers execute and a problem notification is sent out.
5	1	WARNING	HARD	No	Service stabilizes in a HARD problem state. Depending on what the notification interval for the service is, another notification might be sent out.
6	1	OK	HARD	Yes	Service experiences a HARD recovery. Event handlers execute and a recovery notification is sent out.
7	1	OK	HARD	No	Service is still OK.
8	1	UNKNOWN	SOFT	Yes	Service is detected as changing to a SOFT non-OK state. Event handlers execute.
9	2	OK	SOFT	Yes	Service experiences a SOFT recovery. Event handlers execute, but notification are not sent, as this wasn't a "real" problem. State type is set HARD and check # is reset to 1 immediately after this happens.
10	1	OK	HARD	No	Service stabilizes in an OK state.

5.10 Time Periods

Abstract

or..."Is This a Good Time?"

5.10.1 Introduction

5.10.2 Precedence in Time Periods

Timeperod *definitions* may contain multiple types of directives, including weekdays, days of the month, and calendar dates. Different types of directives have different precedence levels and may override other directives in your timeperiod definitions. The order of precedence for different types of directives (in descending order) is as follows:

- Calendar date (2008-01-01)
- Specific month date (January 1st)
- Generic month date (Day 15)
- Offset weekday of specific month (2nd Tuesday in December)
- Offset weekday (3rd Monday)
- Normal weekday (Tuesday)

Examples of different timeperiod directives can be found *here*.

5.10.3 How Time Periods Work With Host and Service Checks

Host and service definitions have an optional "check_period" directive that allows you to specify a timeperiod that should be used to restrict when regularly scheduled, active checks of the host or service can be made.

If you do not use the "check_period directive" to specify a timeperiod, Shinken will be able to schedule active checks of the host or service anytime it needs to. This is essentially a 24x7 monitoring scenario.

Specifying a timeperiod in the "check_period directive" allows you to restrict the time that Shinken perform regularly scheduled, active checks of the host or service. When Shinken attempts to reschedule a host or service check, it will make sure that the next check falls within a valid time range within the defined timeperiod. If it doesn't, Shinken will adjust the next check time to coincide with the next "valid" time in the specified timeperiod. This means that the host or service may not get checked again for another hour, day, or week, etc.

On-demand checks and passive checks are not restricted by the timeperiod you specify in the "check_period directive". Only regularly scheduled active checks are restricted.

A service's timeperiod is inherited from its host only if it's not already defined. In a new shinken installation, it's defined to "24x7" in generic-service. If you want service notifications stopped when the host is outside its notification period, you'll want to comment "notification_period" and/or "notification_enabled" in templates.cfg:generic-service 1.

Unless you have a good reason not to do so, I would recommend that you monitor all your hosts and services using timeperiods that cover a 24x7 time range. If you don't do this, you can run into some problems during "blackout" times (times that are not valid in the timeperiod definition):

- The status of the host or service will appear unchanged during the blackout time.
- Contacts will mostly likely not get re-notified of problems with a host or service during blackout times.
- If a host or service recovers during a blackout time, contacts will not be immediately notified of the recovery.

5.10.4 How Time Periods Work With Contact Notifications

By specifying a timeperiod in the "notification_period" directive of a host or service definition, you can control when Shinken is allowed to send notifications out regarding problems or recoveries for that host or service. When a host notification is about to get sent out, Shinken will make sure that the current time is within a valid range in the "notification_period" timeperiod. If it is a valid time, then Shinken will attempt to notify each contact of the problem or recovery.

You can also use timeperiods to control when notifications can be sent out to individual contacts. By using the "service_notification_period" and "host_notification_period" directives in *contact definitions*, you're able to essentially define an "on call" period for each contact. Contacts will only receive host and service notifications during the times you specify in the notification period directives.

Examples of how to create timeperiod definitions for use for on-call rotations can be found *here*.

5.10.5 How Time Periods Work With Notification Escalations

Service and host *Notification Escalations* have an optional escalation_period directive that allows you to specify a timeperiod when the escalation is valid and can be used. If you do not use the "escalation_period" directive in an escalation definition, the escalation is considered valid at all times. If you specify a timeperiod in the "escalation_period" directive, Shinken will only use the escalation definition during times that are valid in the timeperiod definition.

5.10.6 How Time Periods Work With Dependencies

Host and Service Dependencies have an optional "dependency_period" directive that allows you to specify a timeperiod when the dependendies are valid and can be used. If you do not use the "dependency_period" directive in a dependency definition, the dependency can be used at any time. If you specify a timeperiod in the "dependency_period" directive, Shinken will only use the dependency definition during times that are valid in the timeperiod definition.

5.11 Determining Status and Reachability of Network Hosts

5.11.1 Introduction

If you've ever work in tech support, you've undoubtedly had users tell you "the Internet is down". As a techie, you're pretty sure that no one pulled the power cord from the Internet. Something must be going wrong somewhere between the user's chair and the Internet.

Assuming its a technical problem, you begin to search for the problem. Perhaps the user's computer is turned off, maybe their network cable is unplugged, or perhaps your organization's core router just took a dive. Whatever the problem might be, one thing is most certain - the Internet isn't down. It just happens to be unreachable for that user.

Shinken is able to determine whether the hosts you're monitoring are in a DOWN or UNREACHABLE state. These are very different (although related) states and can help you quickly determine the root cause of network problems. To achieve this goal you must first and foremost define a check_command for the host you are monitoring. From there, here's how the reachability logic works to distinguish between these two states...

5.11.2 Example Network

Take a look at the simple network diagram below. For this example, let us assume you're monitoring all the hosts (server, routers, switches, etc) that are pictured, meaning you have defined check_commands for each of the various hosts. Shinken is installed and running on the Shinken host.

If you have not defined a check_command for your host, Shinken will assume that the host is always UP. Meaning that the logic described will NOT kick-in.

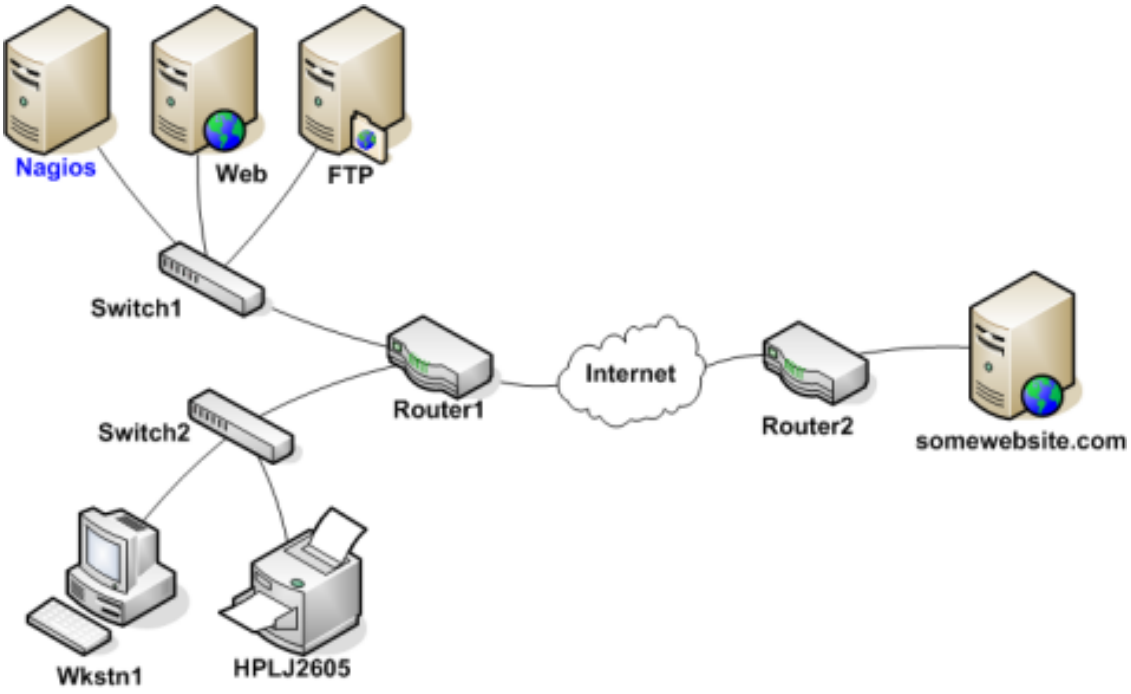

5.11.3 Defining Parent/Child Relationships

In order for Shinken to be able to distinguish between DOWN and UNREACHABLE states for the hosts that are being monitored, you'll need to tell Shinken how those hosts are connected to each other - from the standpoint of the Shinken daemon. To do this, trace the path that a data packet would take from the Shinken daemon to each individual host. Each switch, router, and server the packet encounters or passes through is considered a "hop" and will require that you define a parent/child host relationship in Shinken. Here's what the host parent/child relationships look like from the viewpoint of Shinken:

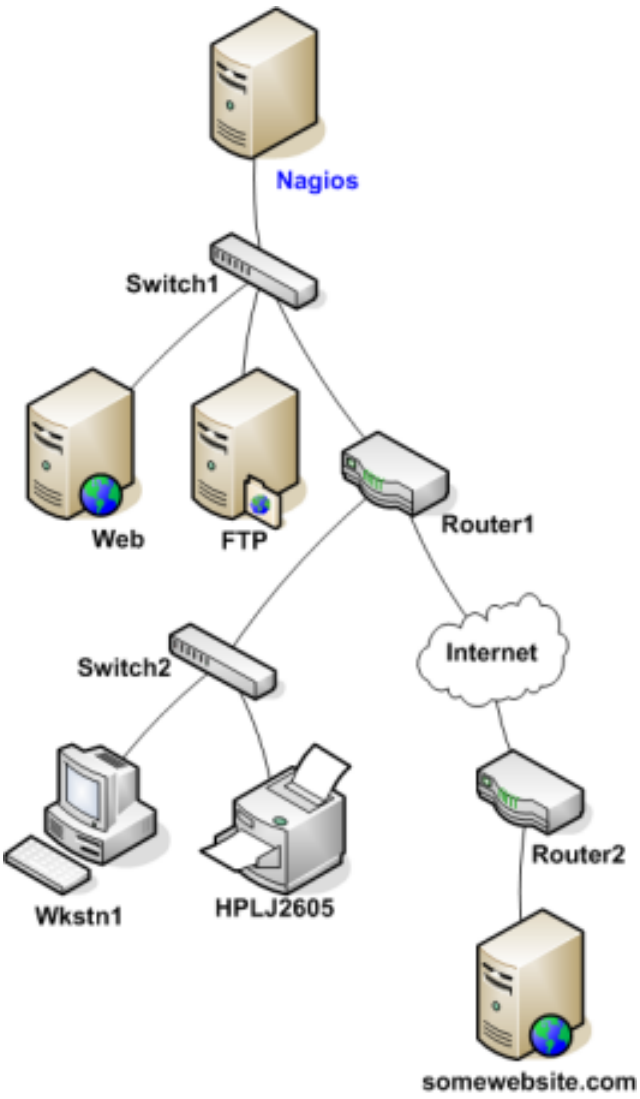

Now that you know what the parent/child relationships look like for hosts that are being monitored, how do you configure Shinken to reflect them? The parents directive in your *host definitions* allows you to do this. Here's what the (abbreviated) host definitions with parent/child relationships would look like for this example:

```
define host{
  host_name    Shinken ; <-- The local host has no parent - it is the topmost host
}

define host{
  host_name    Switch1
  parents    Shinken
}

define host{
  host_name    Web
  parents    Switch1
}

define host{
  host_name    FTP
  parents    Switch1
```

```
}

define host{
  host_name    Router1
  parents    Switch1
}

define host{
  host_name    Switch2
  parents    Router1
}

define host{
  host_name    Wkstn1
  parents    Switch2
}

define host{
  host_name    HPLJ2605
  parents    Switch2
}

define host{
  host_name    Router2
  parents    Router1
}

define host{
  host_name    somewebsite.com
  parents    Router2
}
```

5.11.4 Reachability Logic in Action

Now that you're configured Shinken with the proper parent/child relationships for your hosts, let's see what happen when problems arise. Assume that two hosts - Web and Router1 - go offline...

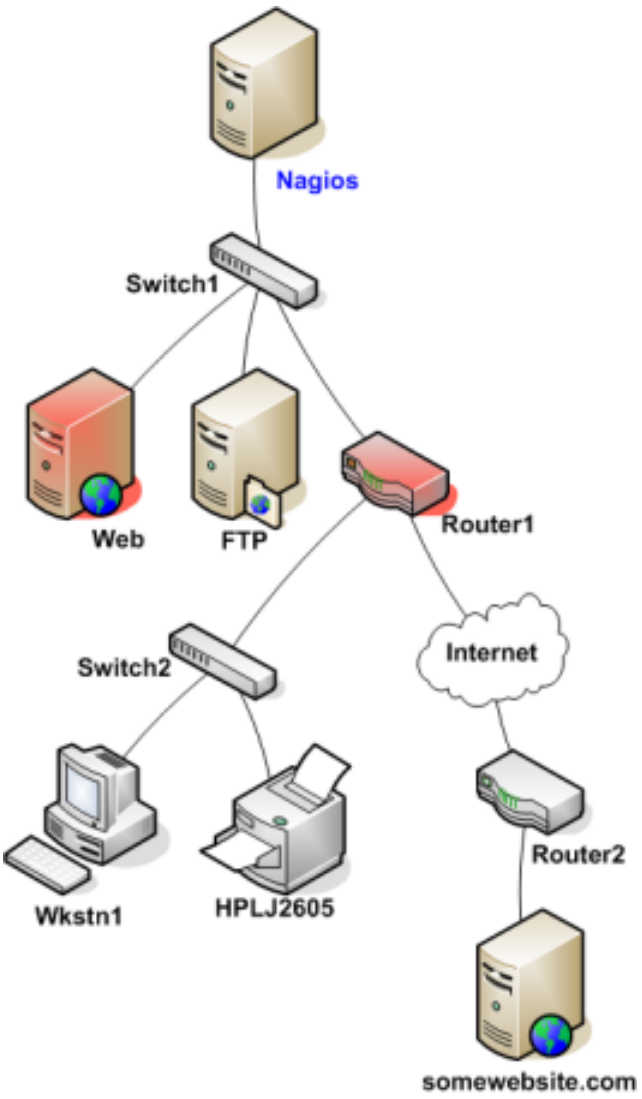

When hosts change state (i.e. from UP to DOWN), the host reachability logic in Shinken kicks in. The reachability logic will initiate parallel checks of the parents and children of whatever hosts change state. This allows Shinken to quickly determine the current status of your network infrastructure when changes occur.

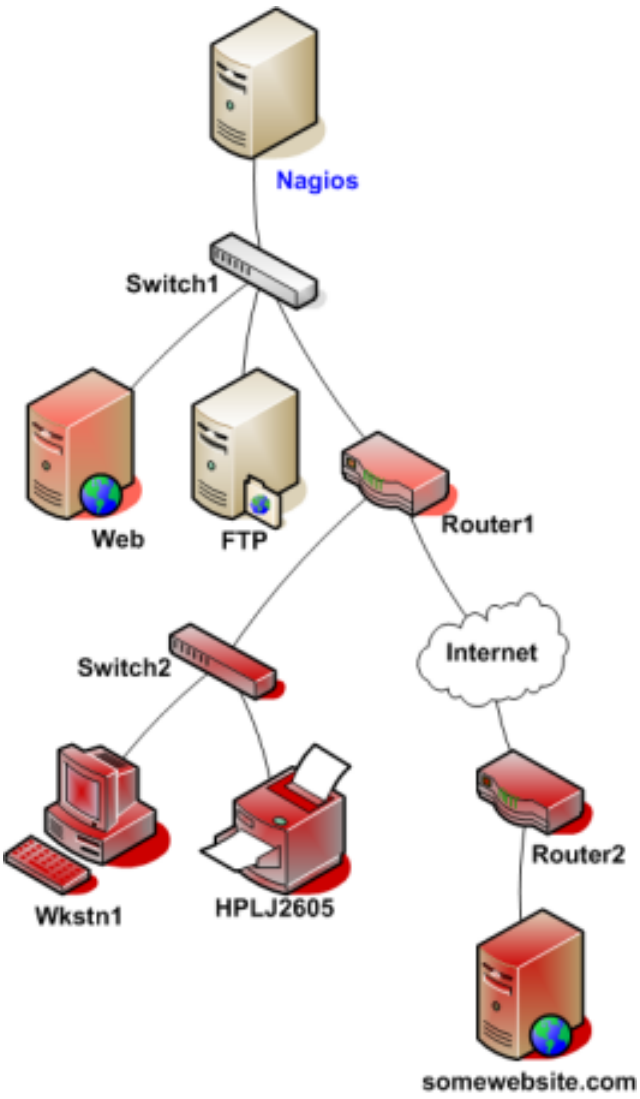

In this example, Shinken will determine that Web and Router1 are both in DOWN states because the "path" to those hosts is not being blocked.

Shinken will determine that all the hosts "beneath" Router1 are all in an UNREACHABLE state because Shinken can't reach them. Router1 is DOWN and is blocking the path to those other hosts. Those hosts might be running fine, or they might be offline - Shinken doesn't know because it can't reach them. Hence Shinken considers them to be UNREACHABLE instead of DOWN.

5.11.5 UNREACHABLE States and Notifications

By default, Shinken will notify contacts about both DOWN and UNREACHABLE host states. As an admin/tech, you might not want to get notifications about hosts that are UNREACHABLE. You know your network structure, and if Shinken notifies you that your router/firewall is down, you know that everything behind it is unreachable.

If you want to spare yourself from a flood of UNREACHABLE notifications during network outages, you can exclude the unreachable (u) option from the "notification_options" directive in your *host* definitions and/or the "host_notification_options" directive in your *contact* definitions.

5.12 Notifications

5.12.1 Introduction

Notification escalations are explained *here*.

5.12.2 When Do Notifications Occur?

The decision to send out notifications is made in the service check and host check logic. Host and service notifications occur in the following instances...

- When a hard state change occurs. More information on state types and hard state changes can be found *here*.
- When a host or service remains in a hard non-OK state and the time specified by the "notification_interval" option in the host or service definition has passed since the last notification was sent out (for that specified host or service).

5.12.3 Who Gets Notified?

Each host and service definition has a "contact_groups" option that specifies what contact groups receive notifications for that particular host or service. Contact groups can contain one or more individual contacts.

When Shinken sends out a host or service notification, it will notify each contact that is a member of any contact groups specified in the "contact_groups" option of the service definition. Shinken realizes that a contact may be a member of more than one contact group, so it removes duplicate contact notifications before it does anything.

5.12.4 What Filters Must Be Passed In Order For Notifications To Be Sent?

Just because there is a need to send out a host or service notification doesn't mean that any contacts are going to get notified. There are several filters that potential notifications must pass before they are deemed worthy enough to be sent out. Even then, specific contacts may not be notified if their notification filters do not allow for the notification to be sent to them. Let's go into the filters that have to be passed in more detail...

5.12.5 Program-Wide Filter:

The first filter that notifications must pass is a test of whether or not notifications are enabled on a program-wide basis. This is initially determined by the *"enable_notifications"* directive in the main config file, but may be changed during runtime from the web interface. If notifications are disabled on a program-wide basis, no host or service notifications can be sent out - period. If they are enabled on a program-wide basis, there are still other tests that must be passed...

5.12.6 Service and Host Filters:

The first filter for host or service notifications is a check to see if the host or service is in a period of *scheduled downtime*. It it is in a scheduled downtime, no one gets notified. If it isn't in a period of downtime, it gets passed on to the next filter. As a side note, notifications for services are suppressed if the host they're associated with is in a period of scheduled downtime.

The second filter for host or service notification is a check to see if the host or service is *flapping* (if you enabled flap detection). If the service or host is currently flapping, no one gets notified. Otherwise it gets passed to the next filter.

The third host or service filter that must be passed is the host- or service-specific notification options. Each service definition contains options that determine whether or not notifications can be sent out for warning states, critical states, and recoveries. Similiarly, each host definition contains options that determine whether or not notifications can be sent out when the host goes down, becomes unreachable, or recovers. If the host or service notification does not pass these options, no one gets notified. If it does pass these options, the notification gets passed to the next filter...

Notifications about host or service recoveries are only sent out if a notification was sent out for the original problem. It doesn't make sense to get a recovery notification for something you never knew was a problem.

The fourth host or service filter that must be passed is the time period test. Each host and service definition has a "notification_period" option that specifies which time period contains valid notification times for the host or service. If the time that the notification is being made does not fall within a valid time range in the specified time period, no one gets contacted. If it falls within a valid time range, the notification gets passed to the next filter...

If the time period filter is not passed, Shinken will reschedule the next notification for the host or service (if its in a non-OK state) for the next valid time present in the time period. This helps ensure that contacts are notified of problems as soon as possible when the next valid time in time period arrives.

The last set of host or service filters is conditional upon two things: (1) a notification was already sent out about a problem with the host or service at some point in the past and (2) the host or service has remained in the same non-OK state that it was when the last notification went out. If these two criteria are met, then Shinken will check and make sure the time that has passed since the last notification went out either meets or exceeds the value specified by the "notification_interval" option in the host or service definition. If not enough time has passed since the last notification, no one gets contacted. If either enough time has passed since the last notification or the two criteria for this filter were not met, the notification will be sent out! Whether or not it actually is sent to individual contacts is up to another set of filters...

5.12.7 Contact Filters:

At this point the notification has passed the program mode filter and all host or service filters and Shinken starts to notify *all the people it should*. Does this mean that each contact is going to receive the notification? No! Each contact has their own set of filters that the notification must pass before they receive it.

Contact filters are specific to each contact and do not affect whether or not other contacts receive notifications.

The first filter that must be passed for each contact are the notification options. Each contact definition contains options that determine whether or not service notifications can be sent out for warning states, critical states, and recoveries. Each contact definition also contains options that determine whether or not host notifications can be sent out when the host goes down, becomes unreachable, or recovers. If the host or service notification does not pass these options, the contact will not be notified. If it does pass these options, the notification gets passed to the next filter...

Notifications about host or service recoveries are only sent out if a notification was sent out for the original problem. It doesn't make sense to get a recovery notification for something you never knew was a problem...

The last filter that must be passed for each contact is the time period test. Each contact definition has a "notification_period" option that specifies which time period contains valid notification times for the contact. If the time that the notification is being made does not fall within a valid time range in the specified time period, the contact will not be notified. If it falls within a valid time range, the contact gets notified!

5.12.8 Notification Methods

You can have Shinken notify you of problems and recoveries pretty much anyway you want: pager, cellphone, email, instant message, audio alert, electric shocker, etc. How notifications are sent depends on the *notification commands* that are defined in your *object definition files*.

If you install Shinken according to the *quickstart guide*, it should be configured to send email notifications. You can see the email notification commands that are used by viewing the contents of the following file: "/usr/local/shinken/etc/objects/commands.cfg".

Specific notification methods (paging, etc.) are not directly incorporated into the Shinken code as it just doesn't make much sense. The "core" of Shinken is not designed to be an all-in-one application. If service checks were embedded in Shinken's core it would be very difficult for users to add new check methods, modify existing checks, etc. Notifications work in a similiar manner. There are a thousand different ways to do notifications and there are already a lot of packages out there that handle the dirty work, so why re-invent the wheel and limit yourself to a bike tire? Its much easier to let an external entity (i.e. a simple script or a full-blown messaging system) do the messy stuff. Some messaging packages that can handle notifications for pagers and cellphones are listed below in the resource section.

5.12.9 Notification Type Macro

When crafting your notification commands, you need to take into account what type of notification is occurring. The *$NOTIFICATIONTYPE$* macro contains a string that identifies exactly that. The table below lists the possible values for the macro and their respective descriptions:

Value	Description
PROBLEM	A service or host has just entered (or is still in) a problem state. If this is a service notification, it means the service is either in a WARNING, UNKNOWN or CRITICAL state. If this is a host notification, it means the host is in a DOWN or UNREACHABLE state.
RECOVERY	A service or host recovery has occurred. If this is a service notification, it means the service has just returned to an OK state. If it is a host notification, it means the host has just returned to an UP state.
ACKNOWLEDGEMENT	This notification is an acknowledgement notification for a host or service problem. Acknowledgement notifications are initiated via the web interface by contacts for the particular host or service.
FLAPPINGSTART	The host or service has just started *flapping*.
FLAPPINGSTOP	The host or service has just stopped *flapping*.
FLAPPINGDISABLED	The host or service has just stopped *flapping* because flap detection was disabled..
DOWNTIMESTART	The host or service has just entered a period of *scheduled downtime*. Future notifications will be supressed.
DOWNTIMESTOP	The host or service has just exited from a period of *scheduled downtime*. Notifications about problems can now resume.
DOWNTIMECANCELLED	The period of *scheduled downtime* for the host or service was just cancelled. Notifications about problems can now resume.

5.12.10 Helpful Resources

There are many ways you could configure Shinken to send notifications out. Its up to you to decide which method(s) you want to use. Once you do that you'll have to install any necessary software and configure notification commands in your config files before you can use them. Here are just a few possible notification methods:

- Email
- Pager
- Phone (SMS)

- WinPopup message
- Yahoo, ICQ, or MSN instant message
- Audio alerts
- etc...

Basically anything you can do from a command line can be tailored for use as a notification command.

If you're looking for an alternative to using email for sending messages to your pager or cellphone, check out these packages. They could be used in conjunction with Shinken to send out a notification via a modem when a problem arises. That way you don't have to rely on email to send notifications out (remember, email may *not* work if there are network problems). I haven't actually tried these packages myself, but others have reported success using them...

- Gnokii (SMS software for contacting Nokia phones via GSM network)
- QuickPage (alphanumeric pager software)
- Sendpage (paging software)

If you want to try out a non-traditional method of notification, you might want to mess around with audio alerts. If you want to have audio alerts played on the monitoring server (with synthesized speech), check out Festival. If you'd rather leave the monitoring box alone and have audio alerts played on another box, check out the Network Audio System (NAS) and rplay projects.

5.13 Active data acquisition modules

5.13.1 Overview

An integrated acquisition module is an optional piece of software that is launched by a Shinken daemon. The module is responsible for doing data acquisition using a specific protocol in a very high performance method. This is preferred over repeatedly calling plugin scripts.

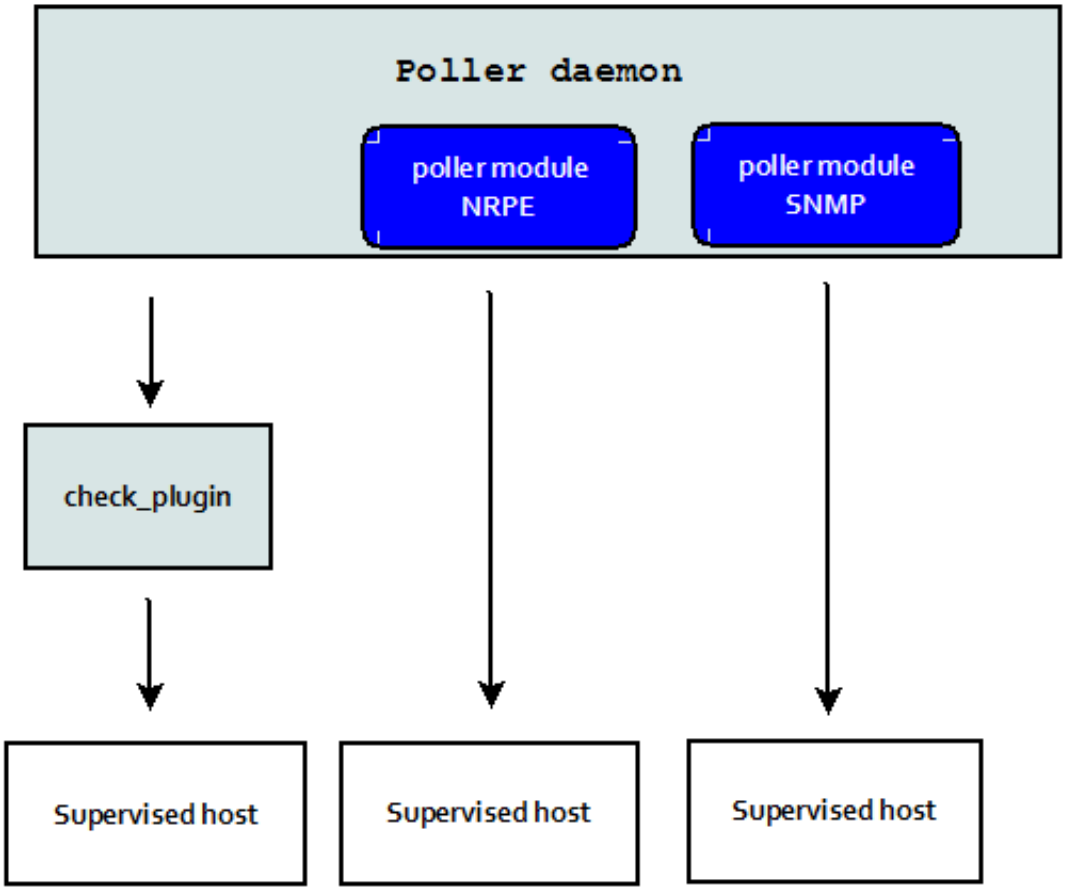

5.13.2 SNMP data acquisition module

Shinken provides an integrated SNMP data acquisition module: SnmpBooster

- *What is the SnmpBooster module*
- *Install and configure the SNMP acquisition module.*
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster Design specification*
- *Reference - SnmpBooster configuration dictionnary*

5.13.3 NRPE data acquisition module

Shinken provides an integrated NRPE data acquisition module. NRPE is a protocol used to communicate with agents installed on remote hosts. It is implemented in the poller daemon to transparently execute NRPE data acquisition. It reads the check command and opens the connection itself. This provides a big performance boost for launching check_nrpe based checks.

The command definitions are identical to the check_nrpe calls.

- *Install and configure the NRPE acquisition module.*

5.13.4 Notes on community Packs

Community provided *monitoring packs* may use the integrated acquisition modules.

Community provided plugins are complimentary to the integrated acquisition modules. **Setup Network and logical dependencies in Shinken**

5.14 Network dependencies

5.14.1 What are network dependencies ?

If you've ever worked in tech support, you've undoubtedly had users tell you "the Internet is down". As a techie, you're pretty sure that no one pulled the power cord from the Internet. Something must be going wrong somewhere between the user's chair and the Internet.

Assuming its a technical problem, you begin to search for the root problem. Perhaps the user's computer is turned off, maybe their network cable is unplugged, or perhaps your organization's core router just took a dive. Whatever the problem might be, one thing is most certain - the Internet isn't down. It just happens to be unreachable for that user.

Shinken is able to determine whether the hosts you're monitoring are in a DOWN or UNREACHABLE state. To do this simply define a check_command for your host. These are very different (although related) states and can help you quickly determine the root cause of network problems. Such dependencies are also possible for applications problems, like your web app is not available because your database is down.

5.14.2 Example Network

Take a look at the simple network diagram below. For this example, lets assume you're monitoring all the hosts (server, routers, switches, etc) that are pictured by defining a check_command for each host. Shinken is installed and running on the Shinken host. **If you have not defined a check_command for your host, Shinken will assume that the host is always UP. Meaning that the logic described will NOT kick-in.**

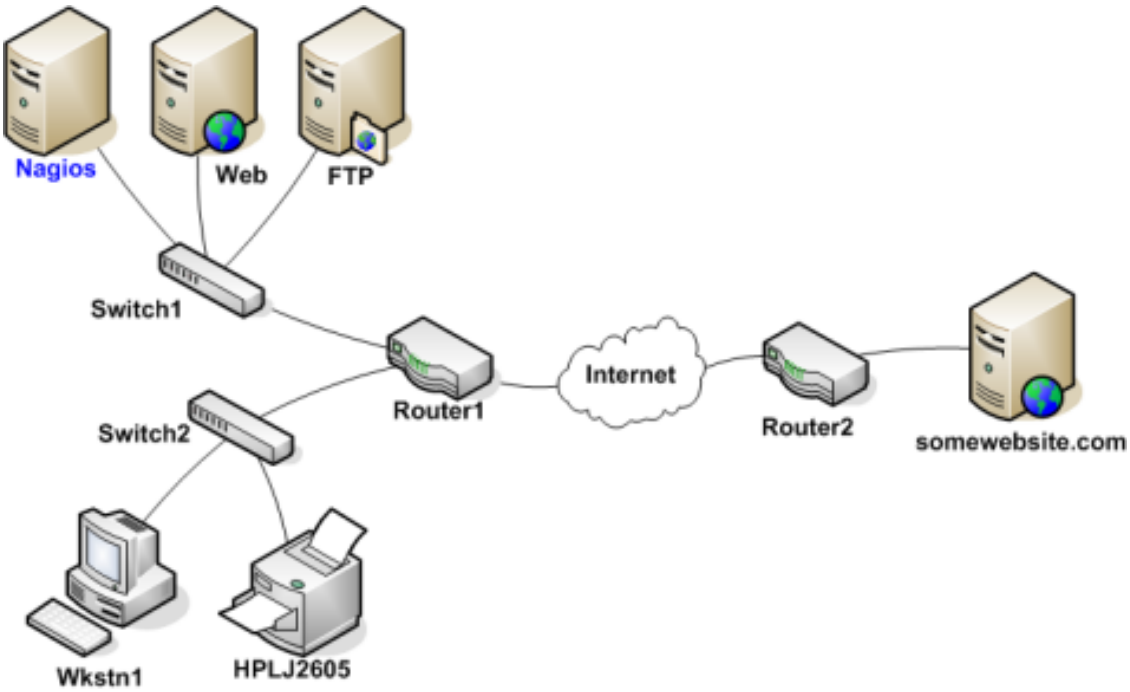

5.14.3 Defining Parent/Child Relationships

The network dependencies will be named "parent/child" relationship. The parent is the switch for example, and the child will be the server.

In order for Shinken to be able to distinguish between DOWN and UNREACHABLE states for the hosts that are being monitored, you'll first need to tell Shinken how those hosts are connected to each other - from the standpoint of the Shinken daemon. To do this, trace the path that a data packet would take from the Shinken daemon to each individual host. Each switch, router, and server the packet encounters or passes through is considered a "hop" and will require that you define a parent/child host relationship in Shinken. Here's what the host parent/child relationships looks like from the viewpoint of Shinken:

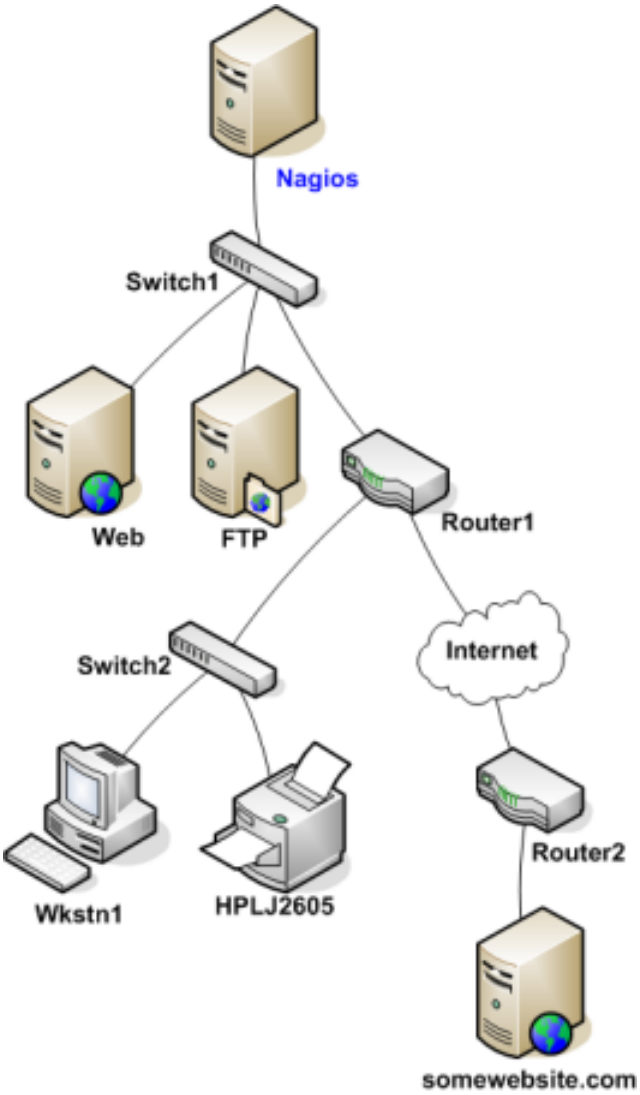

Now that you know what the parent/child relationships look like for hosts that are being monitored, how do you configure Shinken to reflect them? The parents directive in your host definitions allows you to do this. Here's what the (abbreviated) host definitions with parent/child relationships would look like for this example:

```
define host{
  host_name    Shinken ; <-- The local host has no parent - it is the topmost host
}
```

```
define host{
  host_name    Switch1
  parents    Shinken
}

define host{
  host_name    Web
  parents    Switch1
}

define host{
  host_name    FTP
  parents    Switch1
}

define host{
  host_name    Router1
  parents    Switch1
}

define host{
  host_name    Switch2
  parents    Router1
}

define host{
  host_name    Wkstn1
  parents    Switch2
}

define host{
  host_name    HPLJ2605
  parents    Switch2
}

define host{
  host_name    Router2
  parents    Router1
}

define host{
  host_name    somewebsite.com
  parents    Router2
}
```

So basicaly: **in your "child", you declare who is your parent(s)**.

5.14.4 Reachability Logic in Action

Now that you're configured Shinken with the proper parent/child relationships for your hosts, let's see what happen when problems arise. Assume that two hosts - Web and Router1 - go offline...

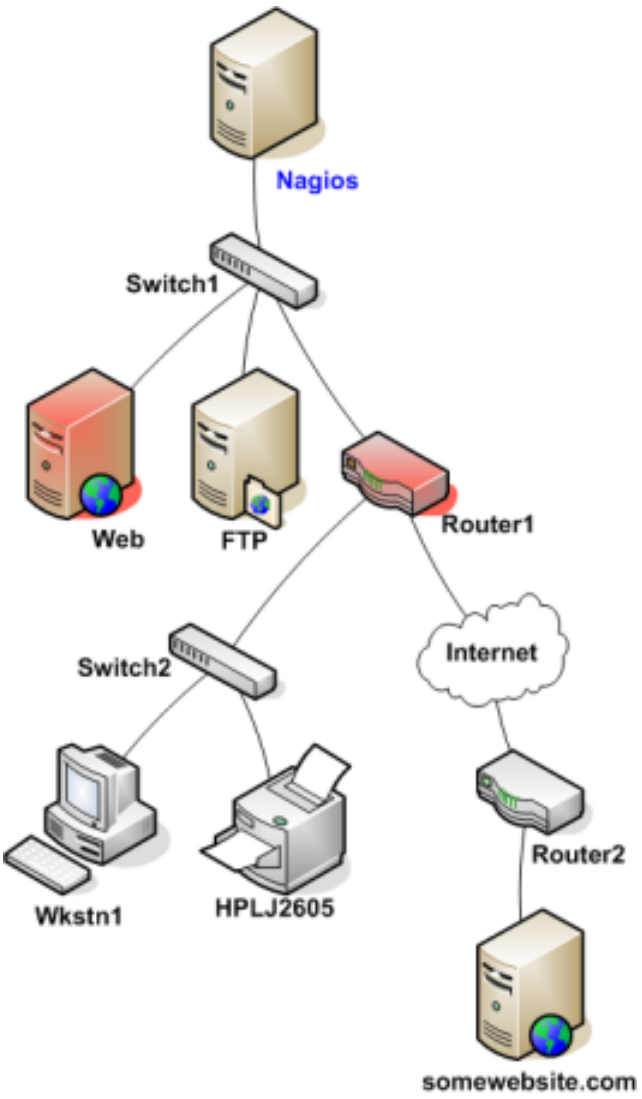

When hosts change state (i.e. from UP to DOWN), the host reachability logic in Shinken kicks in. The reachability logic will initiate parallel checks of the parents and children of whatever hosts change state. This allows Shinken to quickly determine the current status of your network infrastructure when changes occur. During this additonal check time, the notification for the web and router1 hosts are blocked because we don't know yet **WHO** is the root problem.

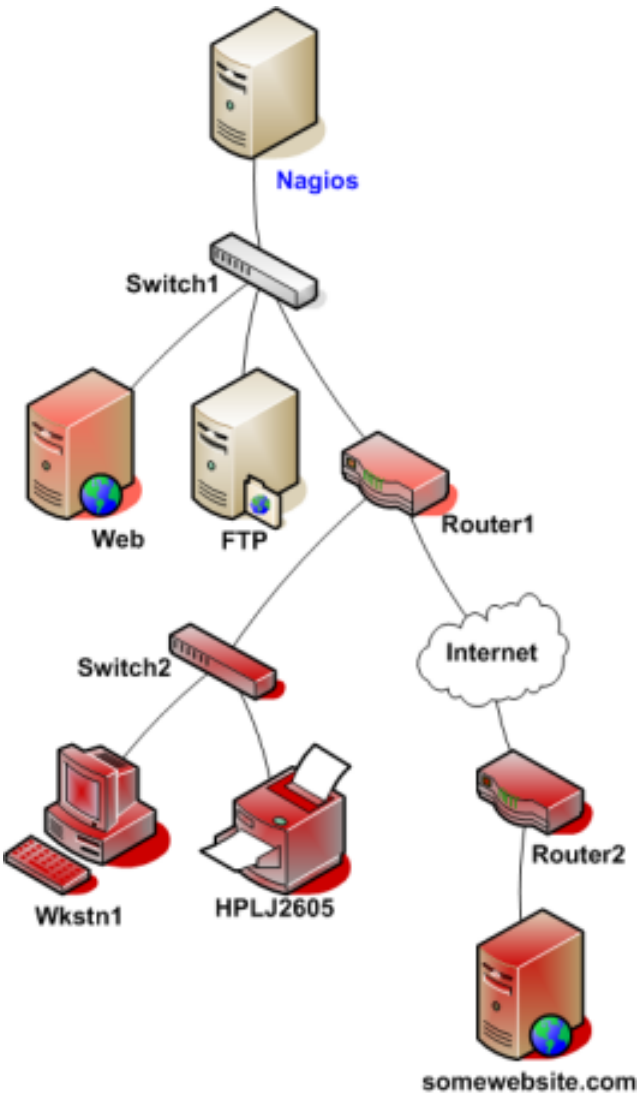

In this example, Shinken will determine that Web and Router1 are both in DOWN states because the "path" to those hosts is not being blocked (switch1 is still alive), and so **it will allow web and router1 notifications to be sent**.

Shinken will determine that all the hosts "beneath" Router1 are all in an UNREACHABLE state because Shinken can't reach them. Router1 is DOWN and is blocking the path to those other hosts. Those hosts might be running fine, or they might be offline - Shinken doesn't know because it can't reach them. Hence Shinken considers them to be UNREACHABLE instead of DOWN, and won't send notifications about them. Such hosts and services beneath router1 are the **impacts** of the **root problem** "router1"

5.14.5 What about more than one parent for an host?

You see that there is a 's' in parents. Because you can define as many parent as you want for an host (like if you got an active/passive switch setup). **The host will be UNREACHABLE only, and only if all it's parents are down or unreachable**. If one is still alive, it will be down. See this as a big *OR* rule.

5.14.6 UNREACHABLE States and Notifications

One important point to remember is **Shinken only notifies about root problems**. If we allow it to notify for root problems AND impacts you will receive too many notifications to quickly find and solve the root problems. That's why Shinken will notify contacts about DOWN hosts, but not for UNREACHABLE ones.

5.14.7 What about notification about services of a down or unreachable hosts?

You will not be notified about all critical or warning errors on a down or unreachable host, because such service states are the impacts of the host root problem. You don't have to configure anything, Shinken will suppress these useless notifications automatically. The official documentation provides more information on *how notifications work*.

5.15 Logical dependencies

Network is not the only element that can have problems. Applications can too.

Service and host dependencies are an advanced feature of Shinken that allows you to control the behavior of hosts and services based on the status of one or more other hosts or services. This section explains how dependencies work, along with the differences between host and service dependencies.

Let's starts with service dependencies. We can take the sample of a Web application service that will depend upon a database service. If the database is failed, it's useless to notify about the web application one, because you already know it's failed. **So Shinken will notify you about your root problem, the database failed, and not about all its impacts, here your web application**.

With only useful notifications, you will be able to find and fix them quickly and not take one hour to find the root problem in your mails.

5.15.1 Service Dependencies Overview

There are a few things you should know about service dependencies:

- A service can be dependent on one or more other services
- A service can be dependent on services which are not associated with the same host
- Advanced service dependencies can be used to cause service check execution and service notifications to be suppressed under different circumstances (OK, WARNING, UNKNOWN, and/or CRITICAL states)
- Advanced service dependencies might only be valid during specific *timeperiods*

5.15.2 Defining simple advanced dependencies

Define a service dependency is quite easy in fact. All you need is to define in your Web application service that it is dependent upon the database service.

```
define service{
  host_name              srv-web
  service_description    Http
  service_dependencies   srv-db,mysql
}
```

So here the web service Http on the host srv-web will depend upon the database service mysql on the host srv-db. If the mysql service has failed, there will be no notifications for service srv-web. If Shinken gets an error state check on the Http service, it will raised a mysql check and suppress the http notification until it knows if the Http service is a root problem or an impact.

5.15.3 Dependencies inheritance

By default, service dependencies are inherited. Let take an example where the mysql service depend upon a nfs service.

```
define service{
  host_name              srv-bd
  service_description    mysql
  service_dependencies   srv-file,nfs,srv-dns,dns
}
```

If Shinken find a problem on Http, it will raise a check on mysql. If this one got a problem too, it will raise a check on the nfs service and srv-dns dns service. If one of these has got a problem too, it will be tagged as the root problem, and will raise a notification for the nfs administrator or dns administrator. If these are ok (dns and nfs), the notification will be sentfor the mysql admin.

5.15.4 And with the host down/unreachable logic?

The dependency logic is done in parallel to the network one. If one logic say it's an impact, then it will tag the problem state as an impact. For example, if the srv-db is down a warning/critical alert on the Http service will be set as an **impact**, like the mysql one, and the root problem will be the srv-bd host that will raise only one notification, a host problem.

5.15.5 Advanced dependencies

For timeperiod limited dependencies or for specific states activation (like for critical states but not warning), please consult the *advanced dependencies* documentation.

5.16 Update Shinken

- grab the latest shinken archive and extract its content
- cd into the resulting folder
- backup shinken configuration plugins and addons and copy the backup id:

```
./install -b</code>
```

Warning: Be careful with your add-ons...Actually Shinken's install script does NOT backs up all add-on configuration files...Take a look at saved files (usually at /opt/backup/bck-shinken.YYYYMMDDhhmmss.tar.gz, need uncompress before search) and check what is and what is not saved before remove. Install script can be easyly improved by adding few lines for other folders at functions "backup" and "restore", see NAGVIS or PNP examples - remove shinken (if you installed addons with the installer say no to the question about removing the addons): <code>./install -u - install the new version:

```
./install -i</code>
- restore the backup: <code>./install -r backupid
```

Important: It's recommended to pull stable version from git. Current master version may be not safe. Please use tagged release. List (not full) of git tags : 1.0rc1, 1.2rc2

CHAPTER 6

Medium

6.1 Business rules

6.1.1 View your infrastructure from a business perspective

The main role of this feature is to allow users to have in one "indicator" the aggregation of other states. This indicator can provide a unique view for users focused on different roles.

Typical roles:

- Service delivery Management
- Business Management
- Engineering
- IT support

Let's take a simple example of a service delivery role for an ERP application. It mainly consists of the following IT components:

- 2 databases, in high availability, so with one database active, the service is considered up
- 2 web servers, in load sharing, so with one web server active, the service is considered up
- 2 load balancers, again in high availability

These IT components (Hosts in this example) will be the basis for the ERP service.

With business rules, you can have an "indicator" representing the "aggregated service" state for the ERP service! Shinken already checks all of the IT components one by one including processing for root cause analysis from a host and service perspective.

6.1.2 How to define Business Rules?

It's a simple service (or a host) with a "special" check_command named bp_rule. :)

Important: Common gotcha: Host status always resolve to either Up, Critical or Unknown. If a host has a warning result from the Host check, it will either make it Critical or Up depending on the Host configuration. A host cannot be in a warning state. Only services can resolve to Up Critical, Warning and Unknown. *Learn more about Host and Service states*

This makes it compatible with all your current habits and UIs. As the service aggregation is considered as any other state from a host or service, you can get notifications, actions and escalations. This means you can have contacts that will receive only the relevant notifications based on their role.

Warning: You do not have to define "bp_rule" command, it's purely internal. You should NOT define it in you checkcommands.cfg file, or the configuration will be invalid due to duplicate commands!

Here is a configuration for the ERP service example, attached to a dummy host named "servicedelivery".

```
define service{
   use         standard-service
   host_name   servicedelivery
   service_description  ERP
   check_command        bp_rule!(h1,database1 | h2,database2) & (h3,Http1 | h4,Http4) & (h5,IPVS1 | h
}
```

That's all!

Note: A complete service delivery view should include an aggregated view of the end user availability perspective states, end user performance perspective states, IT component states, application error states, application performance states. This aggregated state can then be used as a metric for Service Management (basis for defining an SLA).

Warning: From now the business rules manage only one level of () recursivity. Please look at the ticket for more details.

6.1.3 With "need at least X elements" clusters

In some cases, you know that in a cluster of N elements, you need at least X of them to run OK. This is easily defined, you just need to use the "X of:" operator.

Here is an example of the same ERP but with 3 http web servers, and you need at least 2 of them (to handle the load):

```
define service{

   use         standard-service
   host_name   servicedelivery
   service_description  ERP
   check_command        bp_rule!(h1,database1 | h2,database2) & (2 of: h3,Http1 & h4,Http4 & h5,Http5
}
```

It's done :)

6.1.4 The NOT rule

You can define a not state rule. It can be useful for active/passive setups for example. You just need to add a ! before your element name.

Example:

```
define service{

   use         generic-service
   host_name   servicedelivery
   service_description  Cluster_state
   check_command        bp_rule!(h1,database1 & !h2,database2)
}
```

Aggregated state will be okay if database1 is okay and database2 is warning or critical (stopped).

6.1.5 Manage degraded status

In the Xof: way the only case where you got a "warning" (="degraded but not dead") it's when all your elements are in warning. But you should want to be in warning if 1 or your 3 http server is critical: the service is still running, but in a degraded state.

For this you can use the extended operator *X,Y,Zof:*

- X: number min of OK to get an overall OK state
- Y: number min of WARNING to get an overall WARNING state
- Z: number min of CRITICAL to get an overall CRITICAL state

State processing will be done the following order:

- is Ok possible?
- is critical possible?
- is warning possible?
- if none is possible, set OK.

Here are some example for business rules about 5 services A, B, C, D and E. Like 5,1,1of:A|B|C|D|E

Example 1

A	B	C	D	E
Warn	Ok	Ok	Ok	Ok

Rules and overall states:

- 4of: –> Ok
- 5,1,1of: –> Warning
- 5,2,1of: –> Ok

Example 2

A	B	C	D	E
Warn	Warn	Ok	Ok	Ok

Rules and overall states:

- 4of: –> Warning
- 3of: –> Ok
- 4,1,1of: –> Warning

Example 3

A	B	C	D	E
Crit	Crit	Ok	Ok	Ok

Rules and overall states:

- 4of: –> Critical
- 3of: –> Ok
- 4,1,1of: –> Critical

Example 4

A	B	C	D	E
Warn	Crit	Ok	Ok	Ok

Rules and overall states:

- 4of: –> Critical
- 4,1,1of: –> Critical

Example 5

A	B	C	D	E
Warn	Warn	Crit	Ok	Ok

Rules and overall states:

- 2of: –> Ok
- 4,1,1of: –> Critical

Example 6

A	B	C	D	E
Warn	Crit	Crit	Ok	Ok

Rules and overall states:

- 2of: –> Ok * 2,4,4of: –> Ok * 4,1,1of: –> Critical * 4,1,2of: –> Critical * 4,1,3of: –> Warning

Classic cases

Let's look at some classic setups, for MAX elements.

- ON/OFF setup: MAXof: <=> MAX,MAX,MAXof:
- Warning as soon as problem, and critical if all criticals: MAX,1,MAXof:
- Worse state: MAX,1,1

6.2 Monitoring a DMZ

There is two ways for monitoring a DMZ network:

- got a poller on the LAN, and launch check from it, so the firewall should allow monitoring traffic (like nrpe, snmp, etc)
- got a poller on the DMZ, so only the Shinken communications should be open through the firewall

If you can take the first, use it :)

If you can't because your security manager is not happy about it, you should put a poller in the DMZ. So look at the page *Shinken's distributed architecture* first, because you will need a distributed architecture.

Pollers a "dumb" things. They look for jobs to all scheduler (of their realm, if you don't know what is it from now, it's not important). So if you just put a poller in the DMZ network aside another in the LAN, some checks for the dmz will be take by the LAN one, and some for the lan will be take by the DMZ one. It's not a good thing of course :)

6.2.1 Tag your hosts and pollers for being "in the DMZ"

So we will need to "tag" checks, so they will be able to run **only** in the dmz poller, or the lan one.

This tag is done with the poller_tag paramter. It can be applied on the following objects:

- pollers
- commands

- services
- hosts

It's quite simple: you 'tag' objects, and the pollers have got tags too. You've got an implicit inheritance in this order: hosts->services->commands. If a command doesn't have a poller_tag, it will take the one from the service. And if this service doesn't have one neither, it will take the tag from its host.

You just need to install a poller with the 'DMZ' tag in the DMZ and then add it to all hosts (or services) in the DMZ. They will be taken by this poller and you just need to open the port to this poller fom the LAN. Your network admins will be happier :)

6.2.2 Configuration part

So you need to declare in the /etc/shinken.shinken-specific.cfg (or c:shinkenetcshinen-specific.cfg):

```
define poller{

  poller_name    poller-DMZ
  address        server-dmz
  port           7771
  poller_tags    DMZ
}
```

And "tag" some hosts and/or some services.

```
define host{

 host_name  server-DMZ-1
 [...]
 poller_tag DMZ
 [...]
}
```

And that's all :)

All checks for the server-DMZ-1 will be launch from the poller-dmz, and only from it (unless there is another poller in the DMZ with the same tag). you are sure that this check won't be launched from the pollers within the LAN, because untagged pollers can't take tagged checks.

6.3 Shinken High Availability

Shinken makes it easy to have a high availability architecture. Just as easily as the load balancing feature at *Shinken's distributed architecture*

Shinken is business friendly when it comes to meeting availability requirements.

You learned how to add new poller satellites in the *Shinken's distributed architecture*. For the HA the process is the same **You just need to add new satellites in the same way, then define them as "spares".**

You can (should) do the same for all the satellites for a complete HA architecture.

6.3.1 Install all spares daemons on server3

We keep the load balancing of the previous installation and we add a new server (if you do not need load balancing, just take the previous server). This new HA server will be server3 (server2 was for poller load balancing).

So like the previous case, you need to install the daemons but not launch them for now. Look at the 10 min start tutorial to know how to install them on server3.

6.3.2 Declare these spares on server1

Daemons on the server1 neeed to know where their spares are. Everything is done in the main configuration file shinken-specific.cfg. It should be at /etc/shinken/shinken-specific.cfg or c:shinkenetcshinken-specific.cfg).

Add theses lines:

```
define scheduler{

     scheduler_name   scheduler-spare
     address          server3
     port             7768
     spare            1
     }


define poller{

     poller_name     poller-spare
     address         server3
     port            7771
     spare           1
}


define reactionner{

     reactionner_name reactionner-spare
     address          server3
     port             7769
     spare            1
     }


define receiver{

     receiver_name    receiver-spare
     address          server3
     port             7773
     spare            1
}


define broker{

     broker_name     broker-spare
     address         server3
     port            7772
     spare           1
     modules         Simple-log,Livestatus
}


define arbiter{
```

```
    arbiter_name    arbiter-spare
    address         server3
    host_name       server3
    port            7770
    spare           1
}
```

Ok. Configuring HA is defining new daemons on server3 as "spare 1".

WAIT! There are 2 main pitfalls that can halt HA in its tracks:

- Modules - If your master daemon has modules, you must add them on the spare as well!!!!
- Hostname - arbiter-spare has a host_name parameter: it must be the hostname of server3 (so in 99% of the cases, server3). Launch hostname to know the name of your server. If the value is incorrect, the spare arbiter won't start!

6.3.3 Copy all configuration from server1 to server3

Important: It's very important that the two arbiter daemons have the same shinken-specific.cfg file. The whole configuration should also be rsync'ed or copied once a day to ensure the spare arbiter can take over in case of a massive failure of active arbiter.

So copy it in the server3 (overwrite the old one) in the same place.

You do not need to sync all configuration files for hosts and services in the spare. When the master starts, it will synchronize with the spare. But beware, if server1 dies and you must start from fresh on server3, you will not have the full configuration! So synchronize the whole configuration once a day using rsync or other similar method, it is a requirement.

6.3.4 Start :)

Ok, everything is ready. All you need now is to start all the daemons:

```
$server1: sudo /etc/init.d/shinken start
$server3: sudo /etc/init.d/shinken start
```

If an active daemon die, the spare will take over. This is detected in a minute or 2 (you can change it in the shinken-specific.cfg, for each daemon).

Note: For stateful fail-over of a scheduler, link one of the distributed retention modules <distributed retention modules> such as memcache or redis to your schedulers. This will avoid losing the current state of the checks handled by a failed scheduler. Without a retention module, the spare scheduler taking over will need to reschedule all checks and check states will be PENDING until this has completed.

Note: You now have a high availability architecture.

6.4 Mixed GNU/linux AND Windows pollers

There can be as many pollers as you want. And Shinken runs under a lot of systems, like GNU/Linux and Windows. It could be useful to make windows hosts checks *using windows pollers (by a server IN the domain)*, and all the others by a GNU/Linux one (like all pure network based checks).

And in fact you can, and again it's quite easy :) Its important to remember that all pollers connect to all schedulers, so we must have a way to distinguish 'windows' checks from 'gnu/linux' ones.

The poller_tag/poller_tags parameter is useful here. It can be applied on the following objects:

- pollers
- commands
- services
- hosts

It's quite simple: you 'tag' objects, and the pollers have got tags too. You've got an implicit inheritance between hosts->services->commands. If a command doesn't have a poller_tag, it will take the one from the service. And if this service doesn't have one neither, it will take the tag from its host. It's all like the "DMZ" case, but here apply for another purpose.

Let take an example with a 'windows' tag:

```
define command{

 command_name   CheckWMI
 command_line   c:\shinken\libexec\check_wmi.exe -H $HOSTADRESS$ -r $ARG1$
 poller_tag     Windows
}


define poller{

 poller_name  poller-windows
 address      192.168.0.4
 port     7771
 poller_tags  Windows
}
```

And the magic is here: all checks launched with this command will be taken by the poller-windows (or another that has such a tag). A poller with no tags will only take 'untagged' commands.

6.5 Notifications and escalations

6.5.1 Escalations

Shinken supports optional escalation of contact notifications for hosts and services. Escalation of host and service notifications is accomplished by defining escalations and call them from your hosts and services definitions.

Tip: Legacy Nagios host_escalations and service_escalations objects are still managed, but it's adviced to migrate and simplify your configuration with simple escalations objects.

6.5.2 Definition and sample

Notifications are escalated if and only if one or more escalation linked to your host/service matches the current notification that is being sent out. Look at the example below:

```
define escalation{
  escalation_name     To_level_2
  first_notification_time    60
  last_notification_time    240
  notification_interval    60
  contact_groups    level2
}
```

And then you can call it from a service (or an host):

```
define service{
  use                    webservice
  host_name              webserver
  service_description    HTTP
  escalations            To_level_2
  contact_groups         level1
}
```

Here, notifications sent before the fist_notification_time (60 = 60*interval_length*seconds = 60*60s = 1h) will be send to the contact_groups of the service, and between one hour and 4 hours (last_notification_time) it will be escalated to the level2 contact group.

If there is no escalations available (like after 4 hours) it fail back to the default service contact_groups, level1 here.

6.5.3 Lower contact groups

When defining notification escalations, look if it's interesting that were members of "lower" escalations (i.e. those with lower notification time ranges) should also be included in "higher" escalation definitions or not. This can be done to ensure that anyone who gets notified of a problem continues to get notified as the problem is escalated.

In our previous example it becomes:

```
define escalation{
  escalation_name     To_level_2
  first_notification_time    60
  last_notification_time    240
  notification_interval    60
  contact_groups    level1,level2
}
```

6.5.4 Multiple escalations levels

It can be interesting to have more than one level for escalations. Like if problems are send to your level1, and after 1 hour it's send to your level2 and after 4 hours it's send to the level3 until it's resolved.

All you need is to define theses two escalations and link them to your host/service:

```
define escalation{
  escalation_name     To_level_2
  first_notification_time    60
  last_notification_time    240
  notification_interval    60
  contact_groups    level2
}

define escalation{
  escalation_name     To_level_3
  first_notification_time    240
  last_notification_time    0
  notification_interval    60
  contact_groups    level3
}
```

And for your service:

```
define service{
  use                    webservice
  host_name              webserver
  service_description    HTTP
  escalations            To_level_2,To_level_3
  contact_groups         level1
}
```

6.5.5 Overlapping Escalation Ranges

Notification escalation definitions can have notification ranges that overlap. Take the following example:

```
define escalation{
  escalation_name     To_level_2
  first_notification_time    60
  last_notification_time    240
  notification_interval    60
  contact_groups    level2
}

define escalation{
  escalation_name     To_level_3
  first_notification_time    120
  last_notification_time    0
  notification_interval    60
  contact_groups    level3
}
```

In the example above:

- The level2 is notified at one hour
- level 2 and 3 are notified at 2 hours
- Only the level 3 is notified after 4 hours

6.5.6 Recovery Notifications

Recovery notifications are slightly different than problem notifications when it comes to escalations. If the problem was escalated, or was about to reach a new level, who notified for the recovery?

The rule is very simple: we notify about the recovery every one that was notified about the problem, and only them.

6.5.7 Short escalations and long notification Intervals

It's also interesting to see that with escalation, if the notification interval is longer than the next escalation time, it's this last value that will be taken into account.

Let take an example where your service got:

```
define service{
     notification_interval     1440
     escalations    To_level_2,To_level_3
}
```

Then with the escalations objects:

```
define escalation{
  escalation_name   To_level2
  first_notification_time    60
  last_notification_time     120
  contact_groups    level2
}

define escalation{
  escalation_name     To_level_3
  first_notification_time    120
  last_notification_time     0
  contact_groups    level3
}
```

Here let say you have a problem HARD on the service at t=0. It will notify the level1. The next notification should be at t=1440 minutes, so tomorrow. It's okay for classic services (too much notification is DANGEROUS!) but not for escalated ones.

Here, at t=60 minutes, the escalation will raise, you will notify the level2 contact group, and then at t=120 minutes you will notify the level3, and here one a day until they solve it!

So you can put large notification_interval and still have quick escalations times, it's not a problem :)

6.5.8 Time Period Restrictions

Under normal circumstances, escalations can be used at any time that a notification could normally be sent out for the host or service. This "notification time window" is determined by the "notification_period" directive in the host or service definition.

You can optionally restrict escalations so that they are only used during specific time periods by using the "escalation_period" directive in the host or service escalation definition. If you use the "escalation_period" directive to specify a Time Period Definition during which the escalation can be used, the escalation will only be used during that time. If you do not specify any "escalation_period" directive, the escalation can be used at any time within the "notification time window" for the host or service.

Escalated notifications are still subject to the normal time restrictions imposed by the "notification_period" directive in a host or service definition, so the timeperiod you specify in an escalation definition should be a subset of that larger "notification time window".

6.5.9 State Restrictions

If you would like to restrict the escalation definition so that it is only used when the host or service is in a particular state, you can use the "escalation_options" directive in the host or service escalation definition. If you do not use the "escalation_options" directive, the escalation can be used when the host or service is in any state.

6.5.10 Legacy definitions: host_escalations and service_escalations based on notification number

The Nagios legacy escalations definitions are still managed, but it's strongly advice to switch to escalations based on time and call by host/services because it's far more flexible.

Hera are example of theses legacy definitions:

```
define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    3
  last_notification    5
  notification_interval    45
  contact_groups    nt-admins,managers
}

define hostescalation{
  host_name    webserver
  first_notification    6
  last_notification    0
  notification_interval    60
  contact_groups    nt-admins,managers,everyone
}
```

It's based on notification number to know if the escalation should be raised or not. Remember that with this form you cannot mix long notification_interval and short escalations time!

6.6 The Notification Ways, AKA mail 24x7, SMS only the night for a same contact

Let take a classical example: you want email notification 24x7 but SMS only the night and for critical alerts only. How do this with contacts definitions? You have to duplicate the contact. One with notification_period of 24x7 and the email command, and one other with send-by-sms for the night. Duplicate contacts can be hard to manage in services definitions afterward !

That why notification ways are useful: you defined some notification ways (that look really like contacts) and you linked them to your contact.

6.6.1 Example

For example, you can have the below configuration: your contact, a happy admin:

```
define contact{
  contact_name                    happy_admin
  alias                           happy_admin
  email                           admin@localhost
  pager                           +33699999999
  notificationways                email_in_day,sms_the_night
}
```

And now define our notification ways:

```
# Email the whole 24x7 is okay
define notificationway{
     notificationway_name            email_in_day
     service_notification_period     24x7
     host_notification_period        24x7
     service_notification_options    w,u,c,r,f
     host_notification_options       d,u,r,f,s
     service_notification_commands   notify-service
     host_notification_commands      notify-host
}

# But SMS only at night
define notificationway{
     notificationway_name            sms_at_night
     service_notification_period     night
     host_notification_period        night
     service_notification_options    c   ; so only CRITICAL
     host_notification_options       d   ; and DOWN
     service_notification_commands   notify-service-sms
     host_notification_commands      notify-host-sms
}
```

And you can call theses ways from several contacts :)

The contact is valid because he's got valid notificationways. For each notification, we ask for all notification_ways to give us the commands to send. If their notification options or timeperiod is not good, they just do not give one.

6.6.2 Mix old contact with new notification says

Of course, you can still have "old school" contact definition, and even with the notificationways parameters. The service_notification_period parameter of the contact will just be used to create a notificationways like the others, that's all.

6.7 Passive data acquisition

6.7.1 Overview

Definition

Passive check results are received from external source in an unsolicited manner.

Data routing of Passive check results

Passive check results are received by the Arbiter daemon or the Receiver daemon. Modules loaded on either of the daemons permit receiving various types of data formats. The Arbiter is the only one who knows which Scheduler is responsible for the data received. The Arbiter is tasked with administrative blocking functions that can inhibit the responsiveness of the acquisition from a Receiver. This is not an issue for small Shinken installations, as the Arbiter daemon will be blocked for very very short periods of time. In large Shinken installations with tens of thousands of services the Arbiter may induce delays related to passive check result routing.

Shinken 1.2 Shinken uses a direct routing method from the Receiver daemon directly to the appropriate Scheduler for check results and to the Arbiter daemon for external commands.

In all cases, should the Arbiter or Scheduler process be busy doing another task data from the Receiver will **NOT** be lost. It will be queued in the Receiver until the remote daemon can process them.

Configuration Reference

The basics of enabling passive check results in the Shinken configuration

State handling for passive check results

6.7.2 Passive acquisition protocols

NSCA protocol

The NSCA addon consists of a daemon that runs on the Shinken host and a client that is executed from remote hosts. The daemon will listen for connections from remote clients, performs basic validation on the results being submitted, and then write the check results directly into the external command named pipe file (as described in the basics of passive check results configuration) of the Arbiter process or the Receiver process when scaling passive result processing.

The only consideration here is to make sure to configure Shinken Receiver daemons. These will receive the NSCA messages and queue them to be sent to the Arbiter or Scheduler for processing. Benefit from the most common denominator, C compiled cross platform executable, UDP transport, single line output, raw text transmission.

Learn how to configure the NSCA module.

TSCA protocol

Using *TSCA*, you can directly embed in your programs check result submission using a variety of programming languages. Languages such as, but not limited to, Perl, Java and Ruby. Benefit from direct access to Shinken, TCP transport, multi line output and high performance binary transmission.

Learn how to configure the TSCA module.

Shinken WebService protocol

Shinken has its own Python web service module to receive passive data via HTTP(s) using the bottle.py Python module. The Web Service is considered experimental, but is being used in production environments. The ‘WS module ‘_ is very simple and can be extended or improved easily. It is Python after all.

Benefit from direct access to Shinken, HTTP protocol, TCP transport, firewall friendly, multi line output and high performance binary transmission.

Learn how to configure the Shinken WebService.

NSCAweb protocol

A much more evolved protocol for sending data than NSCA. Use curl from the command line to send your data, or submit check results using an HTTP post in your software.

The python NSCAweb listener, http://github.com/smetj/nscaweb, can be hacked to act as a Shinken Receiver module. It might also be possible to wrestle the current Web Service receiver module to process NSCAweb sent messages, the format is the same. .. important:: Should someone be interested in implementing an NSCAweb Shinken Receiver module, support will be provided.

SNMP Traps

Net-SNMP's snmptrapd and SNMP trap translator are typically used to receive, process, and trigger an alerts. Once an alert has been identified an execution is launched of send_nsca, or other method to send result data to a Shinken Receiver daemon. There is no actual Shinken receiver module to receive SNMP traps, but the point is to get the data sent to the Shinken Receiver daemon.

Learn more about SNMP trap handling.

The snmptt documentation has a good writeup on integrating with Nagios, which also applies to Shinken.

There is also a new project by the Check MK team to build an Event console that will process Traps and Syslog messages to create Nagios/Shinken passive check results. It is experimental at this time.

OPC protocol

Various open source and commercial SDKs are available to implement a Shinken Receiver module for getting date from OPC-DA or OPC-UA servers. There is a planned implementations of this module in 2013 for OPC-DA v2 and OPC-UA, but should someone be interested in implementing one, support will be provided.

AMQP protocol

Adding a Shinken Receiver module to act as a consumer of AMQP messages can be implemented without much fuss. There are no planned implementations of this module, but should someone be interested in implementing one, support will be provided. A new broker module for the Canopsis Hypervisor acts as an AMQP endpoint, so this can be used to develop an AMQP consumer or provider. There is also a Python MQ implementation called Krolyk by Jelle Smet that submits check results from AMQP to the Shinken command pipe.

CHAPTER 7

Advanced Topics

7.1 External Commands

7.1.1 Introduction

Shinken can process commands from external applications (including the CGIs and others UIs) and alter various aspects of its monitoring functions based on the commands it receives. External applications can submit commands by writing to the *command file*, which is periodically processed by the Nagios daemon.

7.1.2 Enabling External Commands

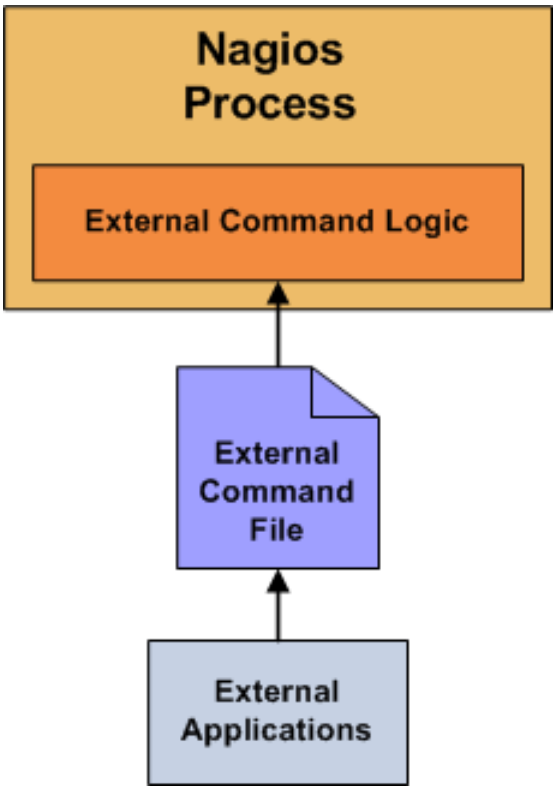

In order to have Shinken process external commands, make sure you do the following:

- Enable external command checking with the *check_external_commands* option.
- Specify the location of the command file with the *command_file* option.
- Setup proper permissions on the directory containing the external command file, as described in the *quickstart guide*.

7.1.3 When Does Shinken Check For External Commands?

In fact every loop it look at it and reap all it can have in the pipe.

7.1.4 Using External Commands

External commands can be used to accomplish a variety of things while Shinken is running. Example of what can be done include temporarily disabling notifications for services and hosts, temporarily disabling service checks, forcing

immediate service checks, adding comments to hosts and services, etc.

7.1.5 Command Format

External commands that are written to the *command file* have the following format...

```
[time] command_id;command_arguments
```

...where time is the time (in "time_t" format) that the external application submitted the external command to the command file. The values for the "command_id" and "command_arguments" arguments will depend on what command is being submitted to Shinken.

A full listing of external commands that can be used (along with examples of how to use them) can be found online at the following URL:

http://www.nagios.org/developerinfo/externalcommands/

7.2 Event Handlers

7.2.1 Introduction

An obvious use for event handlers is the ability for Shinken to proactively fix problems before anyone is notified. Some other uses for event handlers include:

- Restarting a failed service
- Entering a trouble ticket into a helpdesk system
- Logging event information to a database
- Cycling power on a host*
- etc.

Cycling power on a host that is experiencing problems with an automated script should not be implemented lightly. Consider the consequences of this carefully before implementing automatic reboots. :-)

7.2.2 When Are Event Handlers Executed?

Event handlers are executed when a service or host:

- Is in a SOFT problem state
- Initially goes into a HARD problem state
- Initially recovers from a SOFT or HARD problem state

SOFT and HARD states are described in detail *here* .

7.2.3 Event Handler Types

There are different types of optional event handlers that you can define to handle host and state changes:

- Global host event handler
- Global service event handler

- Host-specific event handlers
- Service-specific event handlers

Global host and service event handlers are run for every host or service state change that occurs, immediately prior to any host- or service-specific event handler that may be run.

Event handlers offer functionality similar to notifications (launch some command) but are called each state change, soft or hard. This allows to call handler function and react to problems before Shinken raises a hard state and starts sending out notifications.

You can specify global event handler commands by using the global_host_event_handler and global_service_event_handler options in your main configuration file.

Individual hosts and services can have their own event handler command that should be run to handle state changes. You can specify an event handler that should be run by using the "event_handler" directive in your *host* and *service* definitions. These host- and service-specific event handlers are executed immediately after the (optional) global host or service event handler is executed.

Important: Global event handlers are currently not launched as of April 2013: https://github.com/naparuba/shinken/issues/717

7.2.4 Enabling Event Handlers

Event handlers can be enabled or disabled on a program-wide basis by using the *enable_event_handlers* in your main configuration file.

Host- and service-specific event handlers can be enabled or disabled by using the "event_handler_enabled" directive in your *host* and *service* definitions. Host- and service-specific event handlers will not be executed if the global *enable_event_handlers* option is disabled.

7.2.5 Event Handler Execution Order

As already mentioned, global host and service event handlers are executed immediately before host- or service-specific event handlers.

Event handlers are executed for HARD problem and recovery states immediately after notifications are sent out.

7.2.6 Writing Event Handler Commands

Event handler commands will likely be shell or perl scripts, but they can be any type of executable that can run from a command prompt. At a minimum, the scripts should take the following *macros* as arguments:

For Services: *$SERVICESTATE$*, *$SERVICESTATETYPE$*, *$SERVICEATTEMPT$*

For Hosts: *$HOSTSTATE$*, *$HOSTSTATETYPE$*, *$HOSTATTEMPT$*

The scripts should examine the values of the arguments passed to it and take any necessary action based upon those values. The best way to understand how event handlers work is to see an example. Lucky for you, one is provided *below*.

Additional sample event handler scripts can be found in the "contrib/eventhandlers/" subdirectory of the Nagios distribution. Some of these sample scripts demonstrate the use of *external commands* to implement a *redundant* and *distributed* monitoring environments.

7.2.7 Permissions For Event Handler Commands

Event handler commands will normally execute with the same permissions as the user under which Shinken is running on your machine. This can present a problem if you want to write an event handler that restarts system services, as root privileges are generally required to do these sorts of tasks.

Ideally you should evaluate the types of event handlers you will be implementing and grant just enough permissions to the Shinken user for executing the necessary system commands. You might want to try using sudo to accomplish this.

7.2.8 Service Event Handler Example

The example below assumes that you are monitoring the "HTTP" server on the local machine and have specified restart-httpd as the event handler command for the "HTTP" service definition. Also, I will be assuming that you have set the "max_check_attempts" option for the service to be a value of 4 or greater (i.e. the service is checked 4 times before it is considered to have a real problem). An abbreviated example service definition might look like this...

```
define service{
  host_name    somehost
  service_description    HTTP
  max_check_attempts    4
  event_handler    restart-httpd
  ...
}
```

Once the service has been defined with an event handler, we must define that event handler as a command. An example command definition for restart-httpd is shown below. Notice the macros in the command line that I am passing to the event handler script - these are important!

```
define command{
  command_name    restart-httpd
  command_line    /usr/local/nagios/libexec/eventhandlers/restart-httpd $SERVICESTATE$ $SERVICESTATE
}
```

Now, let's actually write the event handler script (this is the "/usr/local/nagios/libexec/eventhandlers/restart-httpd" script).

```
#!/bin/sh
#
# Event handler script for restarting the web server on the local machine
#
# Note: This script will only restart the web server if the service is
#       retried 3 times (in a "soft" state) or if the web service somehow
#       manages to fall into a "hard" error state.
#
# What state is the HTTP service in?
case "$1" in
OK)
  # The service just came back up, so don't do anything...
  ;;
WARNING)
  # We don't really care about warning states, since the service is probably still running...
  ;;
UNKNOWN)
  # We don't know what might be causing an unknown error, so don't do anything...
  ;;
CRITICAL)
  # Aha!  The HTTP service appears to have a problem - perhaps we should restart the server...
  # Is this a "soft" or a "hard" state?
```

```
  case "$2" in

    # We're in a "soft" state, meaning that Nagios is in the middle of retrying the
    # check before it turns into a "hard" state and contacts get notified...
    SOFT)

    # What check attempt are we on? We don't want to restart the web server on the first
    # check, because it may just be a fluke!
      case "$3" in

        # Wait until the check has been tried 3 times before restarting the web server.
        # If the check fails on the 4th time (after we restart the web server), the state
        # type will turn to "hard" and contacts will be notified of the problem.
        # Hopefully this will restart the web server successfully, so the 4th check will
        # result in a "soft" recovery. If that happens no one gets notified because we
        # fixed the problem!
        3)
          echo -n "Restarting HTTP service (3rd soft critical state)..."
        # Call the init script to restart the HTTPD server
          /etc/rc.d/init.d/httpd restart
          ;;
        esac
        ;;

      # The HTTP service somehow managed to turn into a hard error without getting fixed.
      # It should have been restarted by the code above, but for some reason it didn't.
      # Let's give it one last try, shall we?
      # Note: Contacts have already been notified of a problem with the service at this
      # point (unless you disabled notifications for this service)
      HARD)
        echo -n "Restarting HTTP service..."
        # Call the init script to restart the HTTPD server
        /etc/rc.d/init.d/httpd restart
        ;;
      esac
      ;;
  esac
exit 0
```

The sample script provided above will attempt to restart the web server on the local machine in two different instances:

- After the service has been rechecked for the 3rd time and is in a SOFT CRITICAL state
- After the service first goes into a HARD CRITICAL state

The script should theoretically restart and web server and fix the problem before the service goes into a HARD problem state, but we include a fallback case in the event it doesn't work the first time. It should be noted that the event handler will only be executed the first time that the service falls into a HARD problem state. This prevents Shinken from continuously executing the script to restart the web server if the service remains in a HARD problem state. You don't want that. :-)

That's all there is to it! Event handlers are pretty simple to write and implement, so give it a try and see what you can do.

Note: you may need to:

- disable event handlers during downtimes (either by setting no_event_handlers_during_downtimes=1, or by checking $HOSTDOWNTIME$ and $SERVICEDOWNTIME$)
- make sure you want event handlers to be run even outside of the notification_period

7.3 Volatile Services

7.3.1 Introduction

Shinken has the ability to distinguish between "normal" services and "volatile" services. The is_volatile option in each service definition allows you to specify whether a specific service is volatile or not. For most people, the majority of all monitored services will be non-volatile (i.e. "normal"). However, volatile services can be very useful when used properly...

7.3.2 What Are They Useful For?

Volatile services are useful for monitoring...

- Things that automatically reset themselves to an "OK" state each time they are checked
- Events such as security alerts which require attention every time there is a problem (and not just the first time)

7.3.3 What's So Special About Volatile Services?

Volatile services differ from "normal" services in three important ways. Each time they are checked when they are in a *hard* non-OK state, and the check returns a non-OK state (i.e. no state change has occurred)...

- The non-OK service state is logged
- Contacts are notified about the problem (if that's *what should be done*).Notification intervals are ignored for volatile services.
- The *event handler* for the service is run (if one has been defined)

These events normally only occur for services when they are in a non-OK state and a hard state change has just occurred. In other words, they only happen the first time that a service goes into a non-OK state. If future checks of the service result in the same non-OK state, no hard state change occurs and none of the events mentioned take place again.

Tip: If you are only interested in logging, consider using *stalking* options instead.

7.3.4 The Power Of Two

- *Shinken Configuration*
- *PortSentry Configuration*
- *Port Scan Script*

If you combine the features of volatile services and *passive service checks*, you can do some very useful things. Examples of this include handling "SNMP" traps, security alerts, etc.

How about an example... Let's say you're running PortSentry to detect port scans on your machine and automatically firewall potential intruders. If you want to let Shinken know about port scans, you could do the following...

Shinken Configuration

- Create a service definition called Port Scans and associate it with the host that PortSentry is running on.

- Set the "max_check_attempts" directive in the service definition to 1. This will tell Shinken to immediate force the service into a *hard state* when a non-OK state is reported.
- Set the "active_checks_enabled" directive in the service definition to 0. This prevents Shinken from actively checking the service.
- Set the "passive_checks_enabled" directive in the service definition to 1. This enables passive checks for the service.
- Set this "is_volatile" directive in the service definition to 1.

PortSentry Configuration

Edit your PortSentry configuration file ("portsentry.conf") and define a command for the KILL_RUN_CMD directive as follows:

```
KILL_RUN_CMD="/usr/local/Shinken/libexec/eventhandlers/submit_check_result *"host_name"* 'Port Scans
```

Make sure to replace host_name with the short name of the host that the service is associated with.

Port Scan Script

Create a shell script in the "/usr/local/shinken/libexec/eventhandlers" directory named **submit_check_result**. The contents of the shell script should be something similiar to the following...

```
#!/bin/sh

# Write a command to the Shinken command file to cause
# it to process a service check result

echocmd="/bin/echo"

CommandFile="/usr/local/shinken/var/rw/shinken.cmd"

# get the current date/time in seconds since UNIX epoch
datetime=`date +%s`

# create the command line to add to the command file
cmdline="[$datetime] PROCESS_SERVICE_CHECK_RESULT;$1;$2;$3;$4"

# append the command to the end of the command file
`$echocmd $cmdline >> $CommandFile`
```

What will happen when PortSentry detects a port scan on the machine in the future?

- PortSentry will firewall the host (this is a function of the PortSentry software)
- PortSentry will execute the **submit_check_result** shell script and send a passive check result to Shinken
- Shinken will read the external command file and see the passive service check submitted by PortSentry
- Shinken will put the Port Scans service in a hard CRITICAL state and send notifications to contacts

Pretty neat, huh?

7.4 Service and Host Freshness Checks

7.4.1 Introduction

Shinken supports a feature that does "freshness" checking on the results of host and service checks. The purpose of freshness checking is to ensure that host and service checks are being provided passively by external applications on a regular basis.

Freshness checking is useful when you want to ensure that *passive checks* are being received as frequently as you want. This can be very useful in *distributed* and *failover* monitoring environments.

7.4.2 How Does Freshness Checking Work?

An active check is executed even if active checks are disabled on a program-wide or host- or service-specific basis.

For example, if you have a freshness threshold of 60 for one of your services, Shinken will consider that service to be stale if its last check result is older than 60 seconds.

7.4.3 Enabling Freshness Checking

Here's what you need to do to enable freshness checking...

- Enable freshness checking on a program-wide basis with the *check_service_freshness* and *check_host_freshness* directives.
- Use *service_freshness_check_interval* and *host_freshness_check_interval* options to tell Shinken how often it should check the freshness of service and host results.
- Enable freshness checking on a host- and service-specific basis by setting the "check_freshness" option in your host and service definitions to a value of 1.
- Configure freshness thresholds by setting the "freshness_threshold" option in your host and service definitions.
- Configure the "check_command" option in your host or service definitions to reflect a valid command that should be used to actively check the host or service when it is detected as stale.
- The "check_period" option in your host and service definitions is used when Shinken determines when a host or service can be checked for freshness, so make sure it is set to a valid timeperiod.

If you do not specify a host- or service-specific "freshness_threshold" value (or you set it to zero), Shinken will automatically calculate a threshold automatically, based on a how often you monitor that particular host or service. I would recommended that you explicitly specify a freshness threshold, rather than let Shinken pick one for you.

7.4.4 Example

An example of a service that might require freshness checking might be one that reports the status of your nightly backup jobs. Perhaps you have a external script that submit the results of the backup job to Shinken once the backup is completed. In this case, all of the checks/results for the service are provided by an external application using passive checks. In order to ensure that the status of the backup job gets reported every day, you may want to enable freshness checking for the service. If the external script doesn't submit the results of the backup job, you can have Shinken fake a critical result by doing something like this...

Here's what the definition for the service might look like (some required options are omitted)...

```
define service{
                        host_name               backup-server
                        service_description     ArcServe Backup Job
                        active_checks_enabled   0               ; active checks are NOT enabled
                        passive_checks_enabled  1               ; passive checks are enabled (this is h
                        check_freshness         1
                        freshness_threshold     93600           ; 26 hour threshold, since backups may
                        check_command           no-backup-report        ; this command is run only if t
                        ...other options...
                        }
```

Notice that active checks are disabled for the service. This is because the results for the service are only made by an external application using passive checks. Freshness checking is enabled and the freshness threshold has been set to 26 hours. This is a bit longer than 24 hours because backup jobs sometimes run late from day to day (depending on how much data there is to backup, how much network traffic is present, etc.). The "no-backup-report" command is executed only if the results of the service are determined to be stale. The definition of the "no-backup-report" command might look like this...

```
define command{
                        command_name    no-backup-report
                        command_line    /usr/local/shinken/libexec/check_dummy 2 "CRITICAL: Results of
                        }
```

If Shinken detects that the service results are stale, it will run the "no-backup-report" command as an active service check. This causes the **check_dummy** plugin to be executed, which returns a critical state to Shinken. The service will then go into to a critical state (if it isn't already there) and someone will probably get notified of the problem.

7.5 Distributed Monitoring

7.5.1 Introduction

Shinken can be configured to support distributed monitoring of network services and resources. Shinken is designed for it in contrast to the Nagios way of doing it: which is more of a "MacGyver" way.

7.5.2 Goals

The goal in the distributed monitoring environment is to offload the overhead (CPU usage, etc.) of performing and receiving service checks from a "central" server onto one or more "distributed" servers. Most small to medium sized shops will not have a real need for setting up such an environment. However, when you want to start monitoring thousands of hosts (and several times that many services) using Shinken, this becomes quite important.

7.5.3 The global architecture

Shinken"s architecture has been designed according to the Unix Way: one tool, one task. Shinken has an architecture where each part is isolated and connects to the others via standard interfaces. Shinken is based on the Python Pyro remote objects library. Which makes building a highly available or distributed monitoring architecture quite easy. In contrast, the Nagios daemon does nearly everything: it loads the configuration, schedules and launches checks, and raises notifications.

Major innovations of Shinken over Nagios are to :

- split the different roles into separate daemons

- permit the use of modules to extend and enrich the various Shinken daemons

Shinken core uses **distributed** programming, meaning a daemon will often do remote invocations of code on other daemons, this means that to ensure maximum compatibility and stability, the core language, paths and module versions **must** be the same everywhere a daemon is running.

7.5.4 Shinken Daemon roles

- **Arbiter**: The arbiter daemon reads the configuration, divides it into parts (N schedulers = N parts), and distributes them to the appropriate Shinken daemons. Additionally, it manages the high availability features: if a particular daemon dies, it re-routes the configuration managed by this failed daemon to the configured spare. Finally, it receives input from users (such as external commands from nagios.cmd) or passive check results and routes them to the appropriate daemon. Passive check results are forwarded to the Scheduler responsible for the check. There can only be one active arbiter with other arbiters acting as hot standby spares in the architecture.
  - Modules for data collection: NSCA, TSCA, Ws_arbiter (web service)
  - Modules for configuration data storage: MongoDB,
  - Modules for status retention: PickleRententionArbiter
  - Modules for configuration manipulation: IP_Tag, MySQLImport, GLPI, vmware autolinking and other task specific modules
- **Scheduler**: The scheduler daemon manages the dispatching of checks and actions to the poller and reactionner daemons respectively. The scheduler daemon is also responsible for processing the check result queue, analyzing the results, doing correlation and following up actions accordingly (if a service is down, ask for a host check). It does not launch checks or notifications. It just keeps a queue of pending checks and notifications for other daemons of the architecture (like pollers or reactionners). This permits distributing load equally across many pollers. There can be many schedulers for load-balancing or hot standby roles. Status persistence is achieved using a retention module.
  - Modules for status retention: pickle, nagios, memcache, redis and MongoDB are available.
- **Poller**: The poller daemon launches check plugins as requested by schedulers. When the check is finished it returns the result to the schedulers. Pollers can be tagged for specialized checks (ex. Windows versus Unix, customer A versus customer B, DMZ) There can be many pollers for load-balancing or hot standby spare roles.
  - Module for data acquisition: *NRPE Module*
  - Module for data acquisition: CommandFile (Used for check_mk integration which depends on the nagios.cmd named pipe)
  - Module for data acquisition: SNMPbooster (in development)
- **Reactionner**: The reactionner daemon issues notifications and launches event_handlers. This centralizes communication channels with external systems in order to simplify SMTP authorizations or RSS feed sources (only one for all hosts/services). There can be many reactionners for load-balancing and spare roles * Module for external communications: AndroidSMS
- **Broker**: The broker daemon exports and manages data from schedulers. The management can is done exclusively with modules. Multiple Broker modules can be enabled simultaneously.
  - Module for centralizing Shinken logs: Simple-log (flat file)
  - Modules for data retention: Pickle , ToNdodb_Mysql, ToNdodb_Oracle, couchdb
  - Modules for exporting data: Graphite-Perfdata, NPCDMOD(PNP4Nagios) and Syslog
  - Modules for the Livestatus API - status retention and history: SQLite (default), MongoDB (experimental)
  - Modules for the Shinken WebUI: GRAPHITE_UI, PNP_UI. Trending and data visualization.

  - Modules for compatibility: Service-Perfdata, Host-Perfdata and Status-Dat

- **Receiver** (optional): The receiver daemon receives passive check data and serves as a distributed passive command buffer that will be read by the arbiter daemon. There can be many receivers for load-balancing and hot standby spare roles. The receiver can also use modules to accept data from different protocols. Anyone serious about using passive check results should use a receiver to ensure that when the arbiter is not available (when updating a configuration) all check results are buffered by the receiver and forwarded when the arbiter is back on-line.

  - Module for passive data collection: *NSCA*, *TSCA*, *Ws_arbiter (web service)*

This architecture is fully flexible and scalable: the daemons that require more performance are the poller and the schedulers. The administrator can add as many as he wants. The broker daemon should be on a well provisioned server for larger installations, as only a single broker can be active at one time. A picture is worth a thousand words:

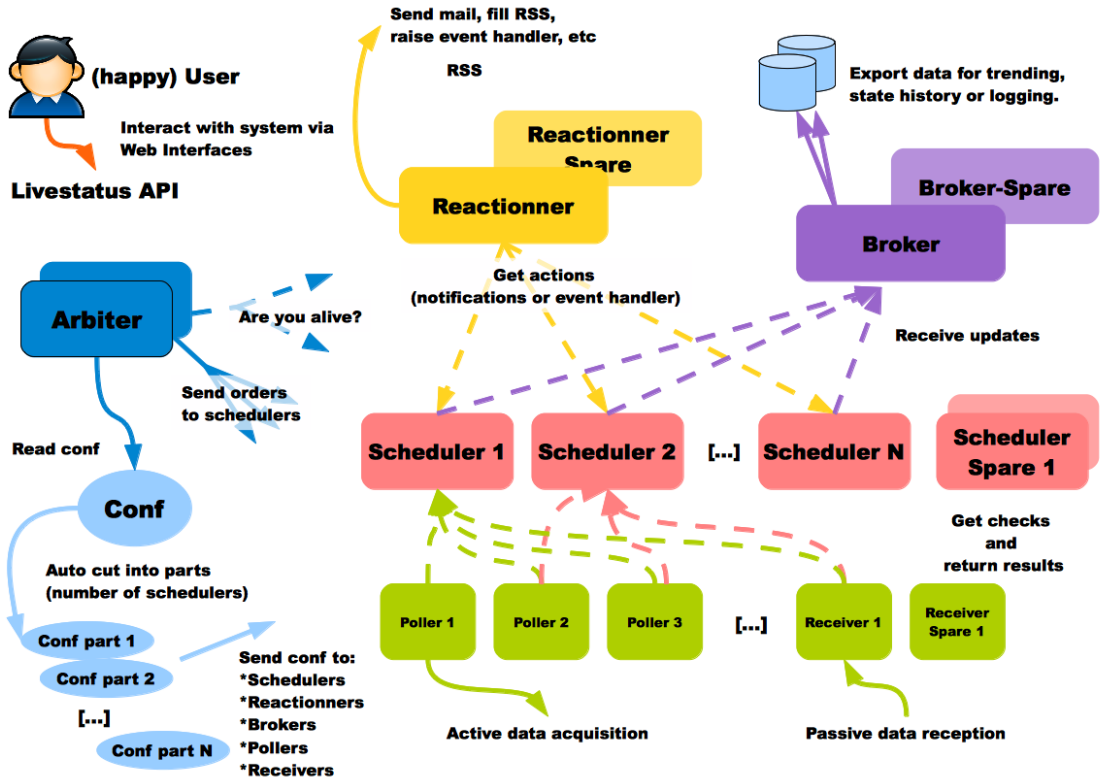

7.5.5 The smart and automatic load balancing

- *Creating independent packs*
- *The packs aggregations into scheduler configurations*
- *The configurations sending to satellites*

Shinken is able to cut the user configuration into parts and dispatch it to the schedulers. The load balancing is done automatically: the administrator does not need to remember which host is linked with another one to create packs, Shinken does it for him.

The dispatch is a host-based one: that means that all services of a host will be in the same scheduler as this host. The major advantage of Shinken is the ability to create independent configurations: an element of a configuration will not have to call an element of another pack. That means that the administrator does not need to know all relations among

elements like parents, hostdependencies or service dependencies: Shinken is able to look at these relations and put these related elements into the same packs.

This action is done in two parts:

- create independent packs of elements
- paste packs to create N configurations for the N schedulers

Creating independent packs

The cutting action is done by looking at two elements: hosts and services. Services are linked with their host so they will be in the same pack. Other relations are taken into account :

- parent relationship for hosts (like a distant server and its router)
- hostdependencies
- servicesdependencies

Shinken looks at all these relations and creates a graph with it. A graph is a relation pack. This can be illustrated by the following picture :

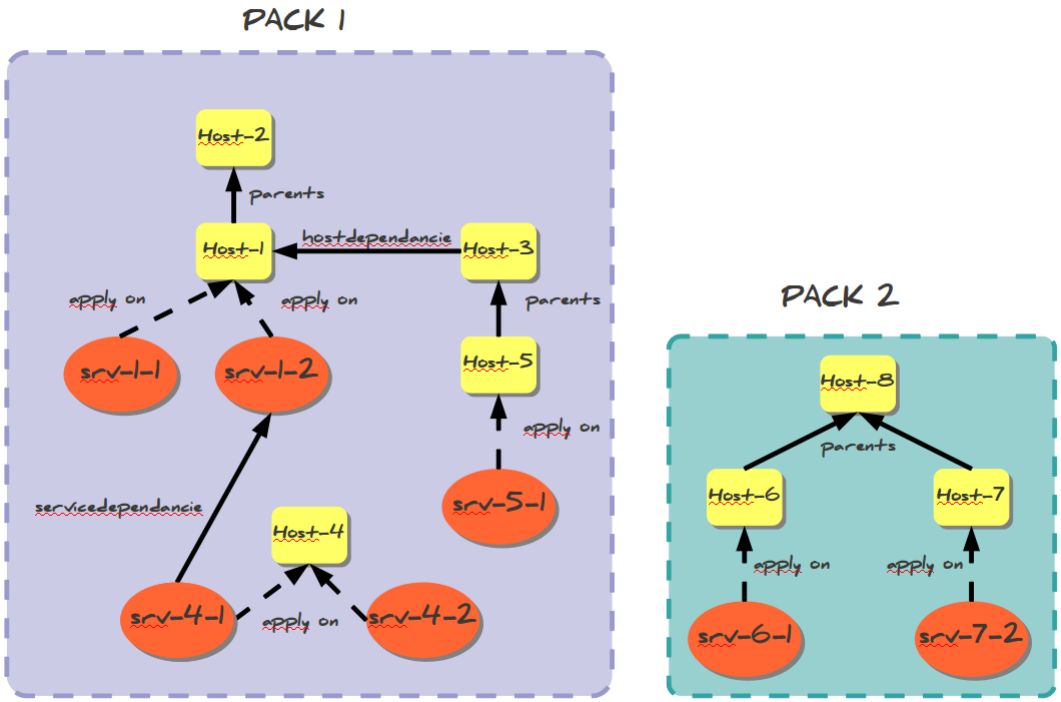

In this example, we will have two packs:

- pack 1: Host-1 to host-5 and all their services
- pack 2: Host-6 to Host-8 and all their services

The packs aggregations into scheduler configurations

When all relation packs are created, the Arbiter aggregates them into N configurations if the administrator has defined N active schedulers (no spares). Packs are aggregated into configurations (it's like "Big packs"). The dispatch looks at the weight property of schedulers: the higher weight a scheduler has, the more packs it will have. This can be shown in the folowing picture :

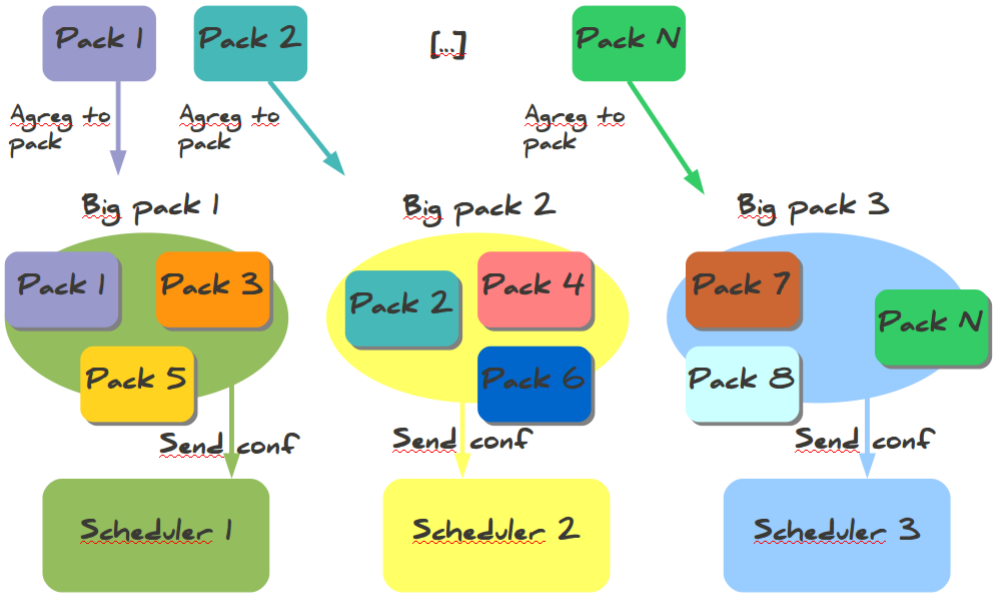

The configurations sending to satellites

When all configurations are created, the Arbiter sends them to the N active Schedulers. A Scheduler can start processing checks once it has received and loaded it's configuration without having to wait for all schedulers to be ready(v1.2). For larger configurations, having more than one Scheduler, even on a single server is highly recommended, as they will load their configurations(new or updated) faster. The Arbiter also creates configurations for satellites (pollers, reactionners and brokers) with links to Schedulers so they know where to get jobs to do. After sending the configurations, the Arbiter begins to watch for orders from the users and is responsible for monitoring the availability of the satellites.

7.5.6 The high availability

- *When a node dies*

The shinken architecture is a high availability one. Before looking at how this works,let's take a look at how the load balancing works if it's now already done.

When a node dies

Nobody is perfect. A server can crash, an application too. That is why administrators have spares: they can take configurations of failing elements and reassign them. For the moment the only daemon that does not have a spare is the Arbiter, but this will be added in the future. The Arbiter regularly checks if everyone is available. If a scheduler or another satellite is dead, it sends its conf to a spare node, defined by the administrator. All satellites are informed by this change so they can get their jobs from the new element and do not try to reach the dead one. If a node was lost due to a network interruption and it comes back up, the Arbiter will notice and ask the old system to drop its configuration. The availability parameters can be modified from the default settings when using larger configurations as the Schedulers or Brokers can become busy and delay their availability responses. The timers are aggressive by default for smaller installations. See shinken-specific configuration parameters for more information on the three timers involved. This can be explained by the following picture :

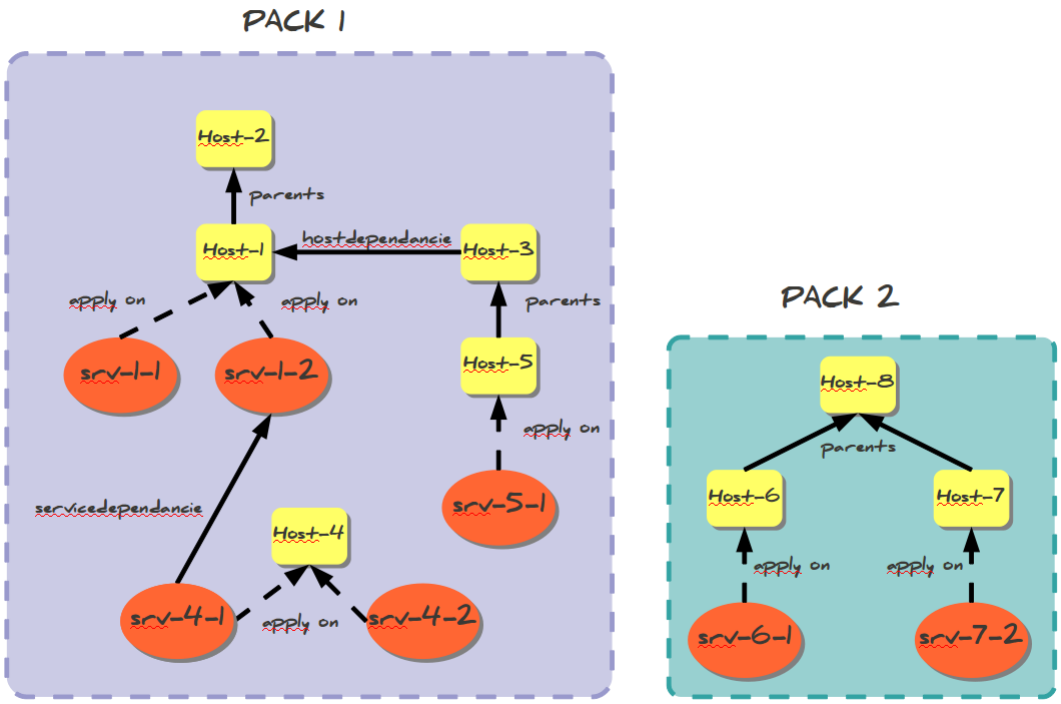

7.5.7 External commands dispatching

The administrator needs to send orders to the schedulers (like a new status for passive checks). In the Shinken way of thinking, the users only need to send orders to one daemon that will then dispatch them to all others. In Nagios the administrator needs to know where the hosts or services are to send the order to the right node. In Shinken the administrator just sends the order to the Arbiter, that's all. External commands can be divided into two types :

- commands that are global to all schedulers
- commands that are specific to one element (host/service).

For each command, Shinken knows if it is global or not. If global, it just sends orders to all schedulers. For specific ones instead it searches which scheduler manages the element referred by the command (host/service) and sends the order to this scheduler. When the order is received by schedulers they just need to apply them.

7.5.8 Different types of Pollers: poller_tag

- *Use cases*

The current Shinken architecture is useful for someone that uses the same type of poller for checks. But it can be useful to have different types of pollers, like GNU/Linux ones and Windows ones. We already saw that all pollers talk to all schedulers. In fact, pollers can be "tagged" so that they will execute only some checks.

This is useful when the user needs to have hosts in the same scheduler (like with dependencies) but needs some hosts or services to be checked by specific pollers (see usage cases below).

These checks can in fact be tagged on 3 levels :

- Host
- Service
- Command

The parameter to tag a command, host or service, is "poller_tag". If a check uses a "tagged" or "untagged" command in a untagged host/service, it takes the poller_tag of this host/service. In a "untagged" host/service, it's the command tag that is taken into account.

The pollers can be tagged with multiple poller_tags. If they are tagged, they will only take checks that are tagged, not the untagged ones, unless they defined the tag "None".

Use cases

This capability is useful in two cases:

- GNU/Linux and Windows pollers
- DMZ

In the first case, it can be useful to have a windows box in a domain with a poller daemon running under a domain account. If this poller launches WMI queries, the user can have an easy Windows monitoring.

The second case is a classic one: when you have a DMZ network, you need to have a dedicated poller that is in the DMZ, and return results to a scheduler in LAN. With this, you can still have dependencies between DMZ hosts and LAN hosts, and still be sure that checks are done in a DMZ-only poller.

7.5.9 Different types of Reactionners: reactionner_tag

- *Use cases*

Like for the pollers, reactionners can also have 'tags'. So you can tag your host/service or commands with "reactionner_tag". If a notification or an event handler uses a "tagged" or "untagged" command in a untagged host/service, it takes the reactionner_tag of this host/service. In a "untaged" host/service, it's the command tag that is taken into account.

The reactionners can be tagged with multiple reactionner_tags. If they are tagged, they will only take checks that are tagged, not the untagged ones, unless they defined the tag "None".

Like for the poller case, it's mainly useful for DMZ/LAN or GNU/Linux/Windows cases.

7.5.10 Advanced architectures: Realms

- *Realms in few words*

- *Realms are not poller_tags!*
- *Sub realms*
- *Example of realm usage*

Shinken's architecture allows the administrator to have a unique point of administration with numerous schedulers, pollers, reactionners and brokers. Hosts are dispatched with their own services to schedulers and the satellites (pollers/reactionners/brokers) get jobs from them. Everyone is happy.

Or almost everyone. Think about an administrator who has a distributed architecture around the world. With the current Shinken architecture the administrator can put a couple scheduler/poller daemons in Europe and another set in Asia, but he cannot "tag" hosts in Asia to be checked by the asian scheduler . Also trying to check an asian server with an european scheduler can be very sub-optimal, read very sloooow. The hosts are dispatched to all schedulers and satellites so the administrator cannot be sure that asian hosts will be checked by the asian monitoring servers.

In the normal Shinken Architecture is useful for load balancing with high availability, for single site.

Shinken provides a way to manage different geographic or organizational sites.

We will use a generic term for this site managment, **Realms**.

Realms in few words

A realm is a pool of resources (scheduler, poller, reactionner and broker) that hosts or hostgroups can be attached to. A host or hostgroup can be attached to only one realm. All "dependancies" or parents of this hosts must be in the same realm. A realm can be tagged "default"' and realm untagged hosts will be put into it. In a realm, pollers, reactionners and brokers will only get jobs from schedulers of the same realm.

Realms are not poller_tags!

Make sure to undestand when to use realms and when to use poller_tags.

- **realms are used to segregate schedulers**
- **poller_tags are used to segregate pollers**

For some cases poller_tag functionality could also be done using Realms. The question you need to ask yourself: Is a poller_tag "enough", or do you need to fully segregate a the scheduler level and use Realms. In realms, schedulers do not communicate with schedulers from other Realms.

If you just need a poller in a DMZ network, use poller_tag.

If you need a scheduler/poller in a customer LAN, use realms.

Sub realms

A realm can contain another realm. It does not change anything for schedulers: they are only responsible for hosts of their realm not the ones of the sub realms. The realm tree is useful for satellites like reactionners or brokers: they can get jobs from the schedulers of their realm, but also from schedulers of sub realms. Pollers can also get jobs from sub realms, but it's less useful so it's disabled by default. Warning: having more than one broker in a scheduler is not a good idea. The jobs for brokers can be taken by only one broker. For the Arbiter it does not change a thing: there is still only one Arbiter and one configuration whatever realms you have.

Example of realm usage

Let's take a look at two distributed environnements. In the first case the administrator wants totally distinct daemons. In the second one he just wants the schedulers/pollers to be distincts, but still have one place to send notifications (reactionners) and one place for database export (broker).

Distincts realms :

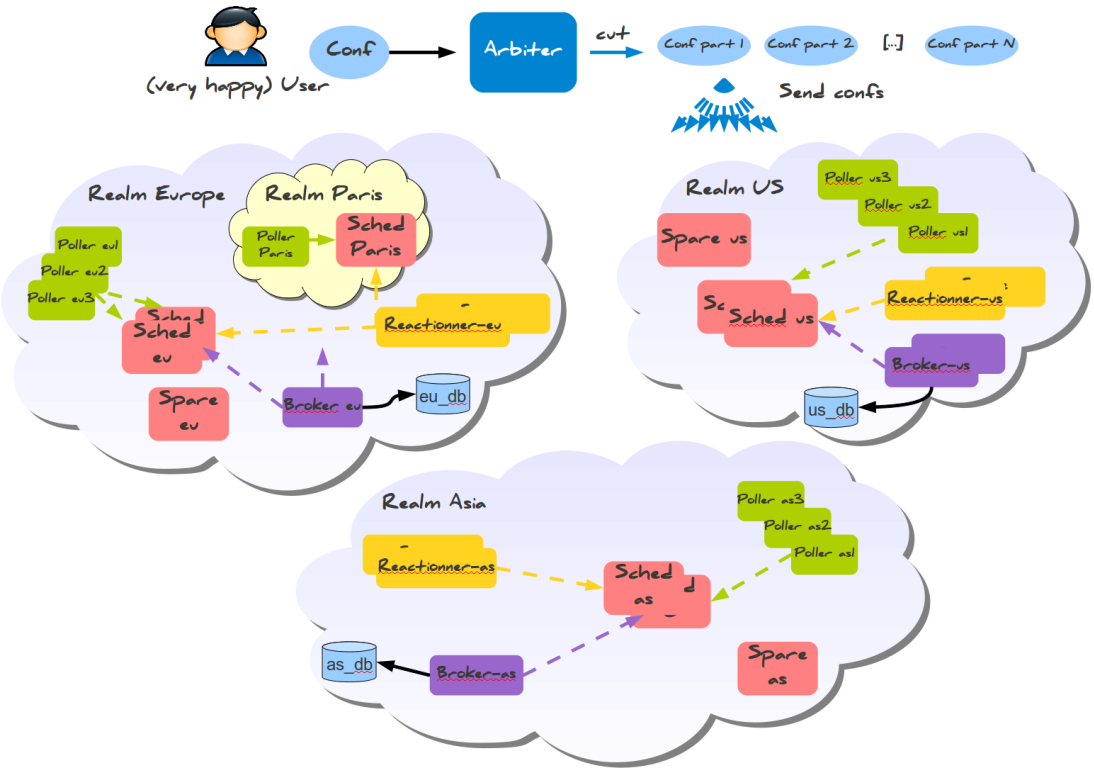

More common usage, the global realm with reactionner/broker, and sub realms with schedulers/pollers :

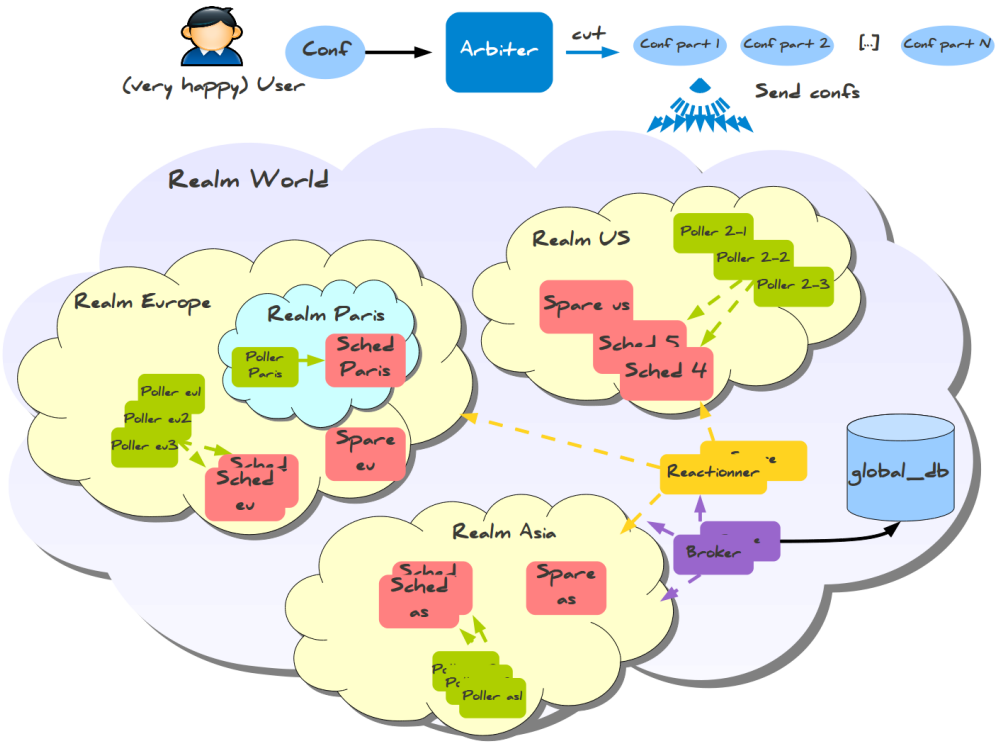

Satellites can be used for their realm or sub realms too. It's just a parameter in the configuration of the element.

7.6 Redundant and Failover Network Monitoring

7.6.1 Introduction

This topic is managed in the distributed section because it's a part of the Shinken architecture.

7.7 Detection and Handling of State Flapping

7.7.1 Introduction

Shinken supports optional detection of hosts and services that are "flapping". Flapping occurs when a service or host changes state too frequently, resulting in a storm of problem and recovery notifications. Flapping can be indicative of configuration problems (i.e. thresholds set too low), troublesome services, or real network problems.

7.7.2 How Flap Detection Works

Before I get into this, let me say that flapping detection has been a little difficult to implement. How exactly does one determine what "too frequently" means in regards to state changes for a particular host or service? When I first started thinking about implementing flap detection I tried to find some information on how flapping could/should be detected. I couldn't find any information about what others were using (where they using any?), so I decided to settle with what seemed to me to be a reasonable solution...

Whenever Shinken checks the status of a host or service, it will check to see if it has started or stopped flapping. It does this by.

- Storing the results of the last 21 checks of the host or service
- Analyzing the historical check results and determine where state changes/transitions occur
- Using the state transitions to determine a percent state change value (a measure of change) for the host or service
- Comparing the percent state change value against low and high flapping thresholds

A host or service is determined to have started flapping when its percent state change first exceeds a high flapping threshold.

A host or service is determined to have stopped flapping when its percent state goes below a low flapping threshold (assuming that is was previously flapping).

7.7.3 Example

Let's describe in more detail how flap detection works with services...

The image below shows a chronological history of service states from the most recent 21 service checks. OK states are shown in green, WARNING states in yellow, CRITICAL states in red, and UNKNOWN states in orange.

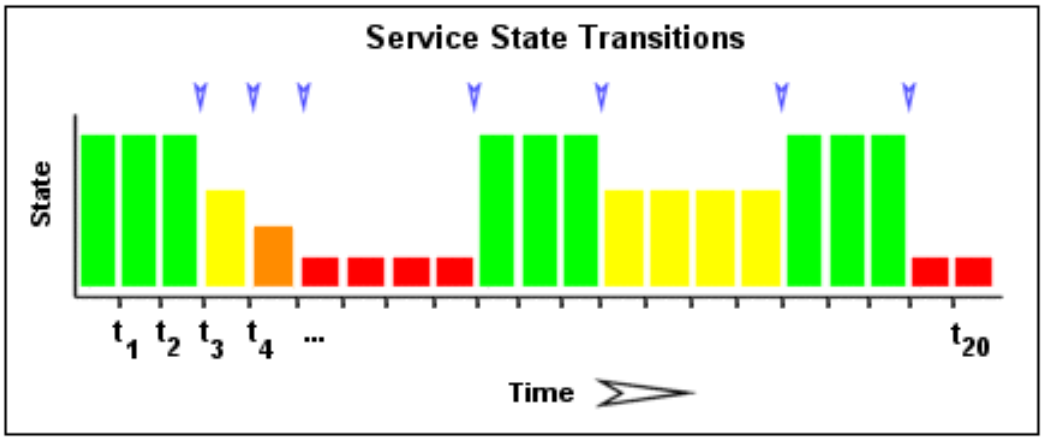

The historical service check results are examined to determine where state changes/transitions occur. State changes occur when an archived state is different from the archived state that immediately precedes it chronologically. Since we keep the results of the last 21 service checks in the array, there is a possibility of having at most 20 state changes. The 20 value can be changed in the main configuration file, see flap_history. In this example there are 7 state changes, indicated by blue arrows in the image above.

The flap detection logic uses the state changes to determine an overall percent state change for the service. This is a measure of volatility/change for the service. Services that never change state will have a 0% state change value, while services that change state each time they're checked will have 100% state change. Most services will have a percent state change somewhere in between.

When calculating the percent state change for the service, the flap detection algorithm will give more weight to new state changes compare to older ones. Specifically, the flap detection routines are currently designed to make the newest possible state change carry 50% more weight than the oldest possible state change. The image below shows how recent state changes are given more weight than older state changes when calculating the overall or total percent state change for a particular service.

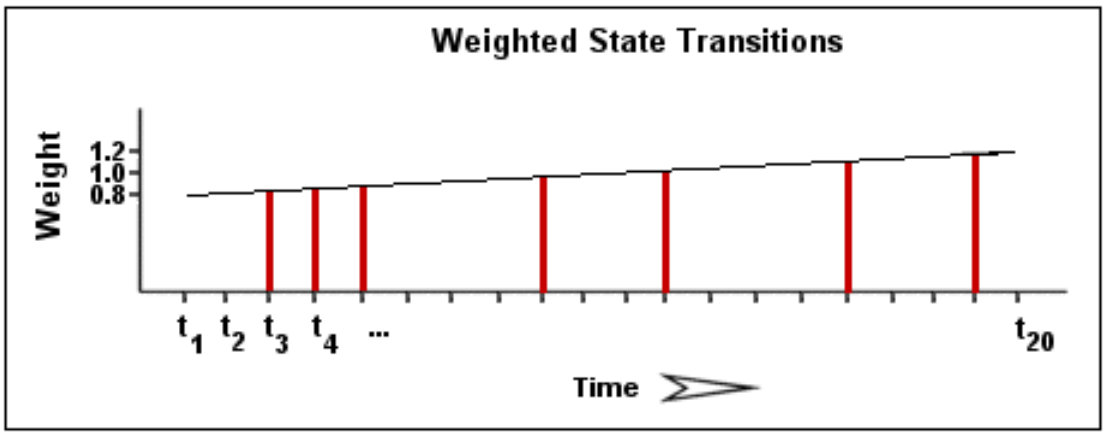

Using the images above, lets do a calculation of percent state change for the service. You will notice that there are a total of 7 state changes (at t3, t4, t5, t9, t12, t16, and t19). Without any weighting of the state changes over time, this would give us a total state change of 35%:

(7 observed state changes / possible 20 state changes) * 100 = 35 %

Since the flap detection logic will give newer state changes a higher rate than older state changes, the actual calculated percent state change will be slightly less than 35% in this example. Let's say that the weighted percent of state change turned out to be 31%...

The calculated percent state change for the service (31%) will then be compared against flapping thresholds to see what should happen:

- If the service was not previously flapping and 31% is equal to or greater than the high flap threshold, Shinken considers the service to have just started flapping.
- If the service was previously flapping and 31% is less than the low flap threshold, Shinken considers the service to have just stopped flapping.

If neither of those two conditions are met, the flap detection logic won't do anything else with the service, since it is either not currently flapping or it is still flapping.

7.7.4 Flap Detection for Services

Shinken checks to see if a service is flapping whenever the service is checked (either actively or passively).

The flap detection logic for services works as described in the example above.

7.7.5 Flap Detection for Hosts

Host flap detection works in a similiar manner to service flap detection, with one important difference: Shinken will attempt to check to see if a host is flapping whenever:

- The host is checked (actively or passively)
- Sometimes when a service associated with that host is checked. More specifically, when at least x amount of time has passed since the flap detection was last performed, where x is equal to the average check interval of all services associated with the host.

Why is this done? With services we know that the minimum amount of time between consecutive flap detection routines is going to be equal to the service check interval. However, you might not be monitoring hosts on a regular basis, so there might not be a host check interval that can be used in the flap detection logic. Also, it makes sense that checking a service should count towards the detection of host flapping. Services are attributes of or things associated

with host after all... At any rate, that's the best method I could come up with for determining how often flap detection could be performed on a host, so there you have it.

7.7.6 Flap Detection Thresholds

Shinken uses several variables to determine the percent state change thresholds is uses for flap detection. For both hosts and services, there are global high and low thresholds and host- or service-specific thresholds that you can configure. Shinken will use the global thresholds for flap detection if you to not specify host- or service- specific thresholds.

The table below shows the global and host- or service-specific variables that control the various thresholds used in flap detection.

Object Type	Global Variables	Object-Specific Variables
Host	*low_host_flap_threshold high_host_flap_threshold*	*low_flap_threshold* *high_flap_threshold*
Service	*low_service_flap_threshold* *high_service_flap_threshold*	*low_flap_threshold* *high_flap_threshold*

7.7.7 States Used For Flap Detection

Normally Shinken will track the results of the last 21 checks of a host or service, regardless of the check result (host/service state), for use in the flap detection logic.

You can exclude certain host or service states from use in flap detection logic by using the "flap_detection_options" directive in your host or service definitions. This directive allows you to specify what host or service states (i.e. "UP, "DOWN", "OK, "CRITICAL") you want to use for flap detection. If you don't use this directive, all host or service states are used in flap detection.

7.7.8 Flap Handling

When a service or host is first detected as flapping, Shinken will:

- Log a message indicating that the service or host is flapping.
- Add a non-persistent comment to the host or service indicating that it is flapping.
- Send a "flapping start" notification for the host or service to appropriate contacts.
- Suppress other notifications for the service or host (this is one of the filters in the *notification logic*).

When a service or host stops flapping, Shinken will:

- Log a message indicating that the service or host has stopped flapping.
- Delete the comment that was originally added to the service or host when it started flapping.
- Send a "flapping stop" notification for the host or service to appropriate contacts.
- Remove the block on notifications for the service or host (notifications will still be bound to the normal *notification logic*).

7.7.9 Enabling Flap Detection

In order to enable the flap detection features in Shinken, you'll need to:

- Set *enable_flap_detection* directive is set to 1.
- Set the "flap_detection_enabled" directive in your host and service definitions is set to 1.

If you want to disable flap detection on a global basis, set the enable_flap_detection directive to 0.

If you would like to disable flap detection for just a few hosts or services, use the "flap_detection_enabled" directive in the host and/or service definitions to do so.

7.8 Notification Escalations

7.8.1 Introduction

Shinken supports optional escalation of contact notifications for hosts and services. Escalation of host and service notifications is accomplished by defining *host escalations* and *service escalations* in your *Object Configuration Overview*.

The examples I provide below all make use of service escalation definitions, but host escalations work the same way. Except, of course, that they're for hosts instead of services. :-)

7.8.2 When Are Notifications Escalated?

Notifications are escalated if and only if one or more escalation definitions matches the current notification that is being sent out. If a host or service notification does not have any valid escalation definitions that applies to it, the contact group(s) specified in either the host group or service definition will be used for the notification. Look at the example below:

```
define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    3
  last_notification    5
  notification_interval    90
  contact_groups    nt-admins,managers
}

define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    6
  last_notification    10
  notification_interval    60
  contact_groups    nt-admins,managers,everyone
}
```

Notice that there are "holes" in the notification escalation definitions. In particular, notifications 1 and 2 are not handled by the escalations, nor are any notifications beyond 10. For the first and second notification, as well as all notifications beyond the tenth one, the default contact groups specified in the service definition are used. For all the examples I'll be using, I'll be assuming that the default contact groups for the service definition is called nt-admins.

7.8.3 Contact Groups

When defining notification escalations, it is important to keep in mind that any contact groups that were members of "lower" escalations (i.e. those with lower notification number ranges) should also be included in "higher" escalation definitions. This should be done to ensure that anyone who gets notified of a problem continues to get notified as the problem is escalated. Example:

```
define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    3
  last_notification    5
  notification_interval    90
  contact_groups    nt-admins,managers
}

define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    6
  last_notification    0
  notification_interval    60
  contact_groups    nt-admins,managers,everyone
}
```

The first (or "lowest") escalation level includes both the nt-admins and managers contact groups. The last (or "highest") escalation level includes the nt-admins, managers, and everyone contact groups. Notice that the nt-admins contact group is included in both escalation definitions. This is done so that they continue to get paged if there are still problems after the first two service notifications are sent out. The managers contact group first appears in the "lower" escalation definition - they are first notified when the third problem notification gets sent out. We want the managers group to continue to be notified if the problem continues past five notifications, so they are also included in the "higher" escalation definition.

7.8.4 Overlapping Escalation Ranges

Notification escalation definitions can have notification ranges that overlap. Take the following example:

```
define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    3
  last_notification    5
  notification_interval    20
  contact_groups    nt-admins,managers
}

define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    4
  last_notification    0
  notification_interval    30
  contact_groups    on-call-support
}
```

In the example above:

- The nt-admins and managers contact groups get notified on the third notification

- All three contact groups get notified on the fourth and fifth notifications
- Only the on-call-support contact group gets notified on the sixth (or higher) notification

7.8.5 Recovery Notifications

Recovery notifications are slightly different than problem notifications when it comes to escalations. Take the following example:

```
define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    3
  last_notification    5
  notification_interval    20
  contact_groups    nt-admins,managers
}

define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    4
  last_notification    0
  notification_interval    30
  contact_groups    on-call-support
}
```

If, after three problem notifications, a recovery notification is sent out for the service, who gets notified? The recovery is actually the fourth notification that gets sent out. However, the escalation code is smart enough to realize that only those people who were notified about the problem on the third notification should be notified about the recovery. In this case, the nt-admins and managers contact groups would be notified of the recovery.

7.8.6 Notification Intervals

You can change the frequency at which escalated notifications are sent out for a particular host or service by using the notification_interval option of the hostgroup or service escalation definition. Example:

```
define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    3
  last_notification    5
  notification_interval    45
  contact_groups    nt-admins,managers
}

define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    6
  last_notification    0
  notification_interval    60
  contact_groups    nt-admins,managers,everyone
}
```

In this example we see that the default notification interval for the services is 240 minutes (this is the value in the service definition). When the service notification is escalated on the 3rd, 4th, and 5th notifications, an interval of 45

minutes will be used between notifications. On the 6th and subsequent notifications, the notification interval will be 60 minutes, as specified in the second escalation definition.

Since it is possible to have overlapping escalation definitions for a particular hostgroup or service, and the fact that a host can be a member of multiple hostgroups, Shinken has to make a decision on what to do as far as the notification interval is concerned when escalation definitions overlap. In any case where there are multiple valid escalation definitions for a particular notification, Shinken will choose the smallest notification interval. Take the following example:

```
define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    3
  last_notification    5
  notification_interval    45
  contact_groups    nt-admins,managers
}

define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    4
  last_notification    0
  notification_interval    60
  contact_groups    nt-admins,managers,everyone
}
```

We see that the two escalation definitions overlap on the 4th and 5th notifications. For these notifications, Shinken will use a notification interval of 45 minutes, since it is the smallest interval present in any valid escalation definitions for those notifications.

One last note about notification intervals deals with intervals of 0. An interval of 0 means that Shinken should only sent a notification out for the first valid notification during that escalation definition. All subsequent notifications for the hostgroup or service will be suppressed. Take this example:

```
define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    3
  last_notification    5
  notification_interval    45
  contact_groups    nt-admins,managers
}

define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    3
  last_notification    5
  notification_interval    45
  contact_groups    nt-admins,managers
}

define serviceescalation{
  host_name    webserver
  service_description    HTTP
  first_notification    7
  last_notification    0
  notification_interval    30
```

```
  contact_groups    nt-admins,managers
}
```

In the example above, the maximum number of problem notifications that could be sent out about the service would be four. This is because the notification interval of 0 in the second escalation definition indicates that only one notification should be sent out (starting with and including the 4th notification) and all subsequent notifications should be repressed. Because of this, the third service escalation definition has no effect whatsoever, as there will never be more than four notifications.

7.8.7 Escalations based on time

The escalations can also be based on time, instead of notification number. It's very easy to setup and work like for the old way but with time instead.

```
define escalation{
  first_notification_time    60
  last_notification_time     120
  contact_groups    nt-admins,managers
}
```

It will use the interval length for the value you set for first/last notification time. Here, it will escalate after 1 hour problem, and stop at 2 hours. You cannot have in the same escalation time and number escalation rules. But of course you can have escalations based on time and escalation based on notification number applied on hosts and services.

7.8.8 Escalations based on time short time

It's also interesting to see that with escalation based on time, if the notification interval is longer than the next escalation time, it's this last value that will be taken into account.

Let take an example where your service got :

```
define service{
     notification_interval     1440
     escalations    ToLevel2,ToLevel3
}
```

Then with the escalations objects :

```
define escalation{
  first_notification_time    60
  last_notification_time     120
  contact_groups    level2
}

  define escalation{
  first_notification_time    120
  last_notification_time     0
  contact_groups    level3
}
```

Here let say you have a problem HARD on the service at t=0. It will notify the level1. The next notification should be at t=1440 minutes, so tomorrow. It's ok for classic services (too much notification is DANGEROUS!) but not for escalated ones.

Here, at t=60 minutes, the escalation will raise, you will notify the level2 contact group, and then at t=120 minutes you will notify the level3, and here one a day until they solve it!

So you can put large notification_interval and still have quick escalations times, it's not a problem :)

7.8.9 Time Period Restrictions

Under normal circumstances, escalations can be used at any time that a notification could normally be sent out for the host or service. This "notification time window" is determined by the "notification_period" directive in the *host* or *service* definition.

You can optionally restrict escalations so that they are only used during specific time periods by using the "escalation_period" directive in the host or service escalation definition. If you use the "escalation_period" directive to specify a *Time Period Definition* during which the escalation can be used, the escalation will only be used during that time. If you do not specify any "escalation_period" directive, the escalation can be used at any time within the "notification time window" for the host or service.

Escalated notifications are still subject to the normal time restrictions imposed by the "notification_period" directive in a host or service definition, so the timeperiod you specify in an escalation definition should be a subset of that larger "notification time window".

7.8.10 State Restrictions

If you would like to restrict the escalation definition so that it is only used when the host or service is in a particular state, you can use the "escalation_options" directive in the host or service escalation definition. If you do not use the "escalation_options" directive, the escalation can be used when the host or service is in any state.

7.9 On-Call Rotations

7.9.1 Introduction

For those lucky admins who have a team of gurus who can help share the responsibility of answering alerts, on-call rotations are often setup. Multiple admins will often alternate taking notifications on weekends, weeknights, holidays, etc.

I'll show you how you can create *timeperiod* definitions in a way that can facilitate most on-call notification rotations. These definitions won't handle human issues that will inevitably crop up (admins calling in sick, swapping shifts, or throwing their pagers into the river), but they will allow you to setup a basic structure that should work the majority of the time.

7.9.2 Scenario 1: Holidays and Weekends

Two admins - John and Bob - are responsible for responding to Shinken alerts. John receives all notifications for weekdays (and weeknights) - except for holidays - and Bob gets handles notifications during the weekends and holidays. Lucky Bob. Here's how you can define this type of rotation using timeperiods...

First, define a timeperiod that contains time ranges for holidays:

```
define timeperiod{
  name    holidays
  timeperiod_name holidays
  january 1    00:00-24:00    ; New Year's Day
  2008-03-23    00:00-24:00    ; Easter (2008)
  2009-04-12    00:00-24:00    ; Easter (2009)
  monday -1 may    00:00-24:00    ; Memorial Day (Last Monday in May)
  july 4    00:00-24:00    ; Independence Day
  monday 1 september    00:00-24:00    ; Labor Day (1st Monday in September)
  thursday 4 november    00:00-24:00    ; Thanksgiving (4th Thursday in November)
  december 25    00:00-24:00    ; Christmas
  december 31    17:00-24:00    ; New Year's Eve (5pm onwards)
 }
```

Next, define a timeperiod for John's on-call times that include weekdays and weeknights, but excludes the dates/times defined in the holidays timeperiod above:

```
define timeperiod{
  timeperiod_name    john-oncall
  monday    00:00-24:00
  tuesday    00:00-24:00
  wednesday    00:00-24:00
  thursday    00:00-24:00
  friday    00:00-24:00
  exclude     holidays    ; Exclude holiday dates/times defined elsewhere
}
```

You can now reference this timeperiod in John's contact definition:

```
define contact{
  contact_name    john
  ...
  host_notification_period    john-oncall
  service_notification_period    john-oncall
}
```

Define a new timeperiod for Bob's on-call times that include weekends and the dates/times defined in the holidays timeperiod above:

```
define timeperiod{
  timeperiod_name    bob-oncall
  friday    00:00-24:00
  saturday    00:00-24:00
  use    holidays    ; Also include holiday date/times defined elsewhere
}
```

You can now reference this timeperiod in Bob's contact definition:

```
define contact{
  contact_name    bob
  ...
  host_notification_period    bob-oncall
  service_notification_period    bob-oncall
}
```

7.9.3 Scenario 2: Alternating Days

In this scenario John and Bob alternate handling alerts every other day - regardless of whether its a weekend, weekday, or holiday.

Define a timeperiod for when John should receive notifications. Assuming today's date is August 1st, 2007 and John is handling notifications starting today, the definition would look like this:

```
define timeperiod{
  timeperiod_name    john-oncall
  2007-08-01 / 2 00:00-24:00    ; Every two days, starting August 1st, 2007
}
```

Now define a timeperiod for when Bob should receive notifications. Bob gets notifications on the days that John doesn't, so his first on-call day starts tomorrow (August 2nd, 2007).

```
define timeperiod{
  timeperiod_name    bob-oncall
  2007-08-02 / 2 00:00-24:00    ; Every two days, starting August 2nd, 2007
}
```

Now you need to reference these timeperiod definitions in the contact definitions for John and Bob:

```
define contact{
  contact_name    john
  ...
  host_notification_period    john-oncall
  service_notification_period    john-oncall
}
define contact{
  contact_name    bob
  ...
  host_notification_period    bob-oncall
  service_notification_period    bob-oncall
}
```

7.9.4 Scenario 3: Alternating Weeks

In this scenario John and Bob alternate handling alerts every other week. John handles alerts Sunday through Saturday one week, and Bob handles alerts for the following seven days. This continues in perpetuity.

Define a timeperiod for when John should receive notifications. Assuming today's date is Sunday, July 29th, 2007 and John is handling notifications this week (starting today), the definition would look like this:

```
define timeperiod{
   timeperiod_name    john-oncall
  2007-07-29 / 14 00:00-24:00    ; Every 14 days (two weeks), starting Sunday, July 29th, 2007
  2007-07-30 / 14 00:00-24:00    ; Every other Monday starting July 30th, 2007
  2007-07-31 / 14 00:00-24:00    ; Every other Tuesday starting July 31st, 2007
  2007-08-01 / 14 00:00-24:00    ; Every other Wednesday starting August 1st, 2007
  2007-08-02 / 14 00:00-24:00    ; Every other Thursday starting August 2nd, 2007
  2007-08-03 / 14 00:00-24:00    ; Every other Friday starting August 3rd, 2007
  2007-08-04 / 14 00:00-24:00    ; Every other Saturday starting August 4th, 2007
}
```

Now define a timeperiod for when Bob should receive notifications. Bob gets notifications on the weeks that John doesn't, so his first on-call day starts next Sunday (August 5th, 2007).

```
define timeperiod{
  timeperiod_name    bob-oncall
  2007-08-05 / 14 00:00-24:00    ; Every 14 days (two weeks), starting Sunday, August 5th, 2007
  2007-08-06 / 14 00:00-24:00    ; Every other Monday starting August 6th, 2007
  2007-08-07 / 14 00:00-24:00    ; Every other Tuesday starting August 7th, 2007
  2007-08-08 / 14 00:00-24:00    ; Every other Wednesday starting August 8th, 2007
  2007-08-09 / 14 00:00-24:00    ; Every other Thursday starting August 9th, 2007
  2007-08-10 / 14 00:00-24:00    ; Every other Friday starting August 10th, 2007
  2007-08-11 / 14 00:00-24:00    ; Every other Saturday starting August 11th, 2007
}
```

Now you need to reference these timeperiod definitions in the contact definitions for John and Bob:

```
define contact{
  contact_name    mjohn
  ...
  host_notification_period    john-oncall
  service_notification_period    john-oncall
}
define contact{
  contact_name    bob
  ...
  host_notification_period    bob-oncall
  service_notification_period    bob-oncall
}
```

7.9.5 Scenario 4: Vacation Days

In this scenarios, John handles notifications for all days except those he has off. He has several standing days off each month, as well as some planned vacations. Bob handles notifications when John is on vacation or out of the office.

First, define a timeperiod that contains time ranges for John's vacation days and days off:

```
define timeperiod{
  name    john-out-of-office
  timeperiod_name    john-out-of-office
  day 15    00:00-24:00    ; 15th day of each month
  day -1    00:00-24:00    ; Last day of each month (28th, 29th, 30th, or 31st)
  day -2    00:00-24:00    ; 2nd to last day of each month (27th, 28th, 29th, or 30th)
  january 2    00:00-24:00    ; January 2nd each year
  june 1 - july 5    00:00-24:00    ; Yearly camping trip (June 1st - July 5th)
  2007-11-01 - 2007-11-10 00:00-24:00    ; Vacation to the US Virgin Islands (November 1st-10th, 2007
}
```

Next, define a timeperiod for John's on-call times that excludes the dates/times defined in the timeperiod above:

```
define timeperiod{
  timeperiod_name    john-oncall
  monday    00:00-24:00
  tuesday    00:00-24:00
  wednesday    00:00-24:00
  thursday    00:00-24:00
  friday    00:00-24:00
  exclude    john-out-of-office    ; Exclude dates/times John is out
}
```

You can now reference this timeperiod in John's contact definition:

```
define contact{
  contact_name    john
  ...
  host_notification_period    john-oncall
  service_notification_period    john-oncall
}
```

Define a new timeperiod for Bob's on-call times that include the dates/times that John is out of the office:

```
define timeperiod{
  timeperod_name    bob-oncall
  use    john-out-of-office    ; Include holiday date/times that John is out
}
```

You can now reference this timeperiod in Bob's contact definition:

```
define contact{
  contact_name    bob
  ...
  host_notification_period    bob-oncall
  service_notification_period    bob-oncall
}
```

7.9.6 Other Scenarios

There are a lot of other on-call notification rotation scenarios that you might have. The date exception directive in *timeperiod definitions* is capable of handling most dates and date ranges that you might need to use, so check out the different formats that you can use. If you make a mistake when creating timeperiod definitions, always err on the side of giving someone else more on-call duty time. :-)

7.10 Monitoring Service and Host Clusters

7.10.1 Introduction

This "cluster" monitoring was managed by the check_cluster2 plugin in the Nagios times, now it's fully integer to the core, so you should read the new business rules that can be used for cluster monitoring :) *Here*.

7.11 Host and Service Dependencies

7.11.1 Introduction

Service and host dependencies are an advanced feature of Shinken that allow you to control the behavior of hosts and services based on the status of one or more other hosts or services. I'll explain how dependencies work, along with the differences between host and service dependencies.

7.11.2 Service Dependencies Overview

There are a few things you should know about service dependencies:

- A service can be dependent on one or more other services

- A service can be dependent on services which are not associated with the same host
- Service dependencies are not inherited (unless specifically configured to)
- Service dependencies can be used to cause service check execution and service notifications to be suppressed under different circumstances (OK, WARNING, UNKNOWN, and/or CRITICAL states)
- Service dependencies might only be valid during specific *timeperiods*

7.11.3 Defining Service Dependencies

First, the basics. You create service dependencies by adding *service dependency definitions* in your *object config file(s)*. In each definition you specify the *dependent* service, the service you are *depending on*, and the criteria (if any) that cause the execution and notification dependencies to fail (these are described later).

You can create several dependencies for a given service, but you must add a separate service dependency definition for each dependency you create.

7.11.4 Example Service Dependencies

The image below shows an example logical layout of service notification and execution dependencies. Different services are dependent on other services for notifications and check execution.

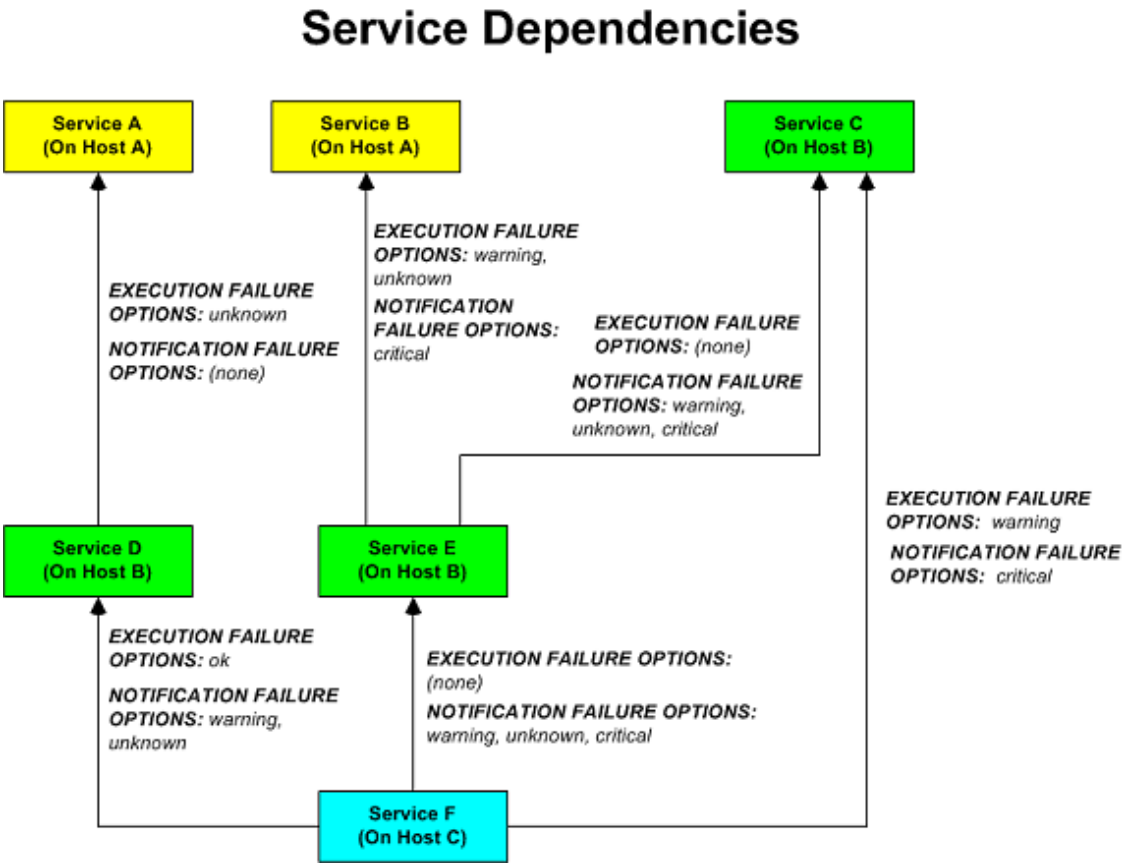

In this example, the dependency definitions for *Service F* on *Host C* would be defined as follows:

```
define servicedependency{
  host_name    Host B
```

```
  service_description    Service D
  dependent_host_name    Host C
  dependent_service_description    Service F
  execution_failure_criteria    o
  notification_failure_criteria    w,u
}

define servicedependency{
  host_name    Host B
  service_description    Service E
  dependent_host_name    Host C
  dependent_service_description    Service F
  execution_failure_criteria    n
  notification_failure_criteria    w,u,c
}

define servicedependency{
  host_name    Host B
  service_description    Service C
  dependent_host_name    Host C
  dependent_service_description    Service F
  execution_failure_criteria    w
  notification_failure_criteria    c
}
```

The other dependency definitions shown in the image above would be defined as follows:

```
define servicedependency{
  host_name    Host A
  service_description    Service A
  dependent_host_name    Host B
  dependent_service_description    Service D
  execution_failure_criteria    u
  notification_failure_criteria    n
}

define servicedependency{
  host_name    Host A
  service_description    Service B
  dependent_host_name    Host B
  dependent_service_description    Service E
  execution_failure_criteria    w,u
  notification_failure_criteria    c
}

define servicedependency{
  host_name    Host B
  service_description    Service C
  dependent_host_name    Host B
  dependent_service_description    Service E
  execution_failure_criteria    n
  notification_failure_criteria    w,u,c
}
```

7.11.5 How Service Dependencies Are Tested

Before Shinken executes a service check or sends notifications out for a service, it will check to see if the service has any dependencies. If it doesn't have any dependencies, the check is executed or the notification is sent out as it normally would be. If the service *does* have one or more dependencies, Shinken will check each dependency entry as follows:

- Shinken gets the current status of the service that is being *depended upon*.
- Shinken compares the current status of the service that is being *depended upon* against either the execution or notification failure options in the dependency definition (whichever one is relevant at the time).
- If the current status of the service that is being *depended upon* matches one of the failure options, the dependency is said to have failed and Shinken will break out of the dependency check loop.
- If the current state of the service that is being *depended upon* does not match any of the failure options for the dependency entry, the dependency is said to have passed and Shinken will go on and check the next dependency entry.

This cycle continues until either all dependencies for the service have been checked or until one dependency check fails.

- One important thing to note is that by default, Shinken will use the most current *hard state* of the service(s) that is/are being depended upon when it does the dependency checks. If you want Shinken to use the most current state of the services (regardless of whether its a soft or hard state), enable the *soft_state_dependencies* option.

7.11.6 Execution Dependencies

Execution dependencies are used to restrict when *active checks* of a service can be performed. *Passive checks* are not restricted by execution dependencies.

If all of the execution dependency tests for the service passed, Shinken will execute the check of the service as it normally would. If even just one of the execution dependencies for a service fails, Shinken will temporarily prevent the execution of checks for that (dependent) service. At some point in the future the execution dependency tests for the service may all pass. If this happens, Shinken will start checking the service again as it normally would. More information on the check scheduling logic can be found *here*.

In the example above, **Service E** would have failed execution dependencies if **Service B** is in a WARNING or UNKNOWN state. If this was the case, the service check would not be performed and the check would be scheduled for (potential) execution at a later time.

7.11.7 Notification Dependencies

If all of the notification dependency tests for the service *passed*, Shinken will send notifications out for the service as it normally would. If even just one of the notification dependencies for a service fails, Shinken will temporarily repress notifications for that (dependent) service. At some point in the future the notification dependency tests for the service may all pass. If this happens, Shinken will start sending out notifications again as it normally would for the service. More information on the notification logic can be found *here*.

In the example above, **Service F** would have failed notification dependencies if **Service C** is in a CRITICAL state, *and/or* **Service D** is in a WARNING or UNKNOWN state, *and/or/* if **Service E** is in a WARNING, UNKNOWN, or CRITICAL state. If this were the case, notifications for the service would not be sent out.

7.11.8 Dependency Inheritance

As mentioned before, service dependencies are not inherited by default. In the example above you can see that Service F is dependent on Service E. However, it does not automatically inherit Service E's dependencies on Service B and Service C. In order to make Service F dependent on Service C we had to add another service dependency definition. There is no dependency definition for Service B, so Service F is not dependent on Service B.

If you do wish to make service dependencies inheritable, you must use the inherits_parent directive in the *service dependency* definition. When this directive is enabled, it indicates that the dependency inherits dependencies of the service that is being depended upon (also referred to as the master service). In other words, if the master service is dependent upon other services and any one of those dependencies fail, this dependency will also fail.

In the example above, imagine that you want to add a new dependency for service F to make it dependent on service A. You could create a new dependency definition that specified service F as the dependent service and service A as being the master service (i.e. the service that is being dependend on). You could alternatively modify the dependency definition for services D and F to look like this:

```
define servicedependency{
  host_name    Host B
  service_description    Service D
  dependent_host_name    Host C
  dependent_service_description    Service F
  execution_failure_criteria    o
  notification_failure_criteria    n
  inherits_parent    1
}
```

Since the inherits_parent directive is enabled, the dependency between services A and D will be tested when the dependency between services F and D are being tested.

Dependencies can have multiple levels of inheritance. If the dependency definition between A and D had its inherits_parent directive enable and service A was dependent on some other service (let's call it service G), the service F would be dependent on services D, A, and G (each with potentially different criteria).

7.11.9 Host Dependencies

As you'd probably expect, host dependencies work in a similar fashion to service dependencies. The difference is that they're for hosts, not services.

Do not confuse host dependencies with parent/child host relationships. You should be using parent/child host relationships (defined with the parents directive in *host* definitions) for most cases, rather than host dependencies. A description of how parent/child host relationships work can be found in the documentation on *network reachability*.

Here are the basics about host dependencies:

- A host can be dependent on one or more other host
- Host dependencies are not inherited (unless specifically configured to)
- Host dependencies can be used to cause host check execution and host notifications to be suppressed under different circumstances (UP, DOWN, and/or UNREACHABLE states)
- Host dependencies might only be valid during specific *timeperiods*

7.11.10 Example Host Dependencies

The image below shows an example of the logical layout of host notification dependencies. Different hosts are dependent on other hosts for notifications.

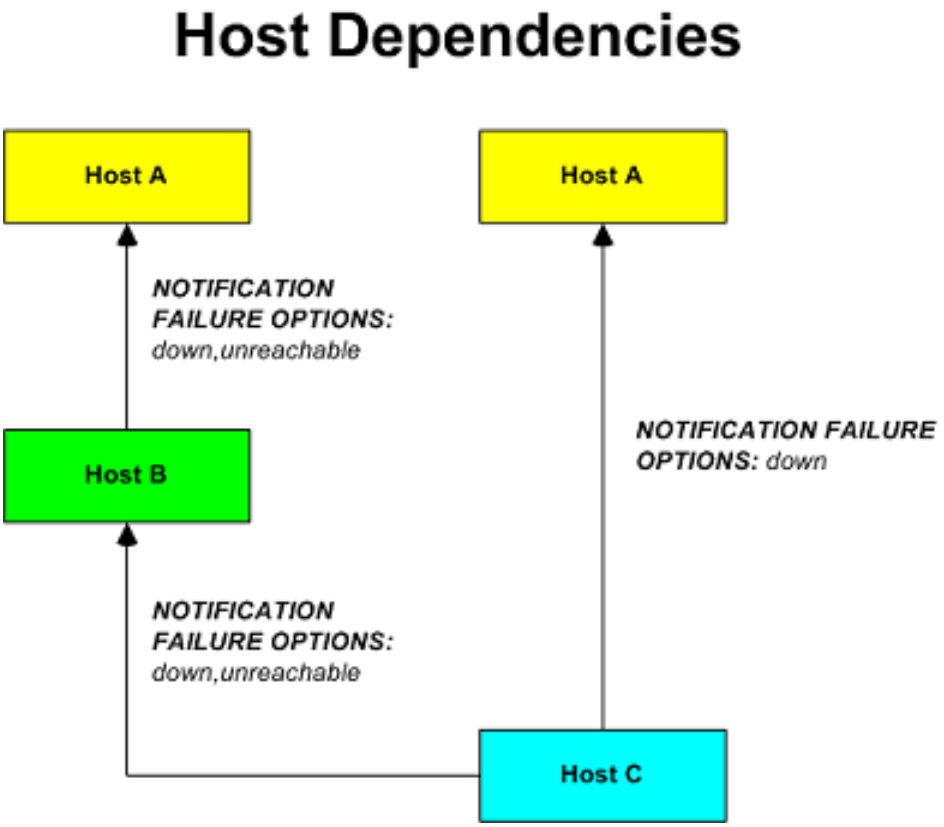

In the example above, the dependency definitions for Host C would be defined as follows:

```
define hostdependency{
  host_name    Host A
  dependent_host_name    Host C
  notification_failure_criteria    d
}

define hostdependency{
  host_name    Host B
  dependent_host_name    Host C
  notification_failure_criteria    d,u
}
```

As with service dependencies, host dependencies are not inherited. In the example image you can see that Host C does not inherit the host dependencies of Host B. In order for Host C to be dependent on Host A, a new host dependency definition must be defined.

Host notification dependencies work in a similar manner to service notification dependencies. If *all* of the notification dependency tests for the host *pass*, Shinken will send notifications out for the host as it normally would. If even just one of the notification dependencies for a host fails, Shinken will temporarily repress notifications for that (dependent) host. At some point in the future the notification dependency tests for the host may all pass. If this happens, Shinken will start sending out notifications again as it normally would for the host. More information on the notification logic can be found *here*.

7.12 State Stalking

7.12.1 Introduction

State "stalking" is a feature which is probably not going to used by most users. When enabled, it allows you to log changes in the output service and host checks even if the state of the host or service does not change. When stalking is enabled for a particular host or service, Shinken will watch that host or service very carefully and log any changes it sees in the output of check results. As you'll see, it can be very helpful to you in later analysis of the log files.

7.12.2 How Does It Work?

Under normal circumstances, the result of a host or service check is only logged if the host or service has changed state since it was last checked. There are a few exceptions to this, but for the most part, that's the rule.

If you enable stalking for one or more states of a particular host or service, Shinken will log the results of the host or service check if the output from the check differs from the output from the previous check. Take the following example of eight consecutive checks of a service:

Given this sequence of checks, you would normally only see two log entries for this catastrophe. The first one would occur at service check x+2 when the service changed from an OK state to a WARNING state. The second log entry would occur at service check x+3 when the service changed from a WARNING state to a CRITICAL state.

For whatever reason, you may like to have the complete history of this catastrophe in your log files. Perhaps to help explain to your manager how quickly the situation got out of control, perhaps just to laugh at it over a couple of drinks at the local pub...

Well, if you had enabled stalking of this service for CRITICAL states, you would have events at x+4 and x+5 logged in addition to the events at x+2 and x+3. Why is this? With state stalking enabled, Shinken would have examined the output from each service check to see if it differed from the output of the previous check. If the output differed and the state of the service didn't change between the two checks, the result of the newer service check would get logged.

A similar example of stalking might be on a service that checks your web server. If the **check_http** plugin first returns a WARNING state because of a 404 error and on subsequent checks returns a WARNING state because of a particular pattern not being found, you might want to know that. If you didn't enable state stalking for WARNING states of the service, only the first WARNING state event (the 404 error) would be logged and you wouldn't have any idea (looking back in the archived logs) that future WARNING states were not due to a 404, but rather some text pattern that could not be found in the returned web page.

7.12.3 Should I Enable Stalking?

First, you must decide if you have a real need to analyze archived log data to find the exact cause of a problem. You may decide you need this feature for some hosts or services, but not for all. You may also find that you only have a need to enable stalking for some host or service states, rather than all of them. For example, you may decide to enable stalking for WARNING and CRITICAL states of a service, but not for OK and UNKNOWN states.

The decision to to enable state stalking for a particular host or service will also depend on the plugin that you use to check that host or service. If the plugin always returns the same text output for a particular state, there is no reason to enable stalking for that state.

7.12.4 How Do I Enable Stalking?

You can enable state stalking for hosts and services by using the stalking_options directive in *host and service definitions*.

7.12.5 How Does Stalking Differ From Volatile Services?

Volatile services are similar, but will cause notifications and event handlers to run. Stalking is purely for logging purposes.

7.12.6 Caveats

You should be aware that there are some potential pitfalls with enabling stalking. These all relate to the reporting functions found in various CGIs (histogram, alert summary, etc.). Because state stalking will cause additional alert entries to be logged, the data produced by the reports will show evidence of inflated numbers of alerts.

As a general rule, I would suggest that you *not* enable stalking for hosts and services without thinking things through. Still, it's there if you need and want it.

7.13 Performance Data

7.13.1 Introduction

Shinken is designed to allow *plugins* to return optional performance data in addition to normal status data, as well as allow you to pass that performance data to external applications for processing. A description of the different types of performance data, as well as information on how to go about processing that data is described below...

7.13.2 Types of Performance Data

There are two basic categories of performance data that can be obtained from Shinken:

- Check performance data
- Plugin performance data

Check performance data is internal data that relates to the actual execution of a host or service check. This might include things like service check latency (i.e. how "late" was the service check from its scheduled execution time) and the number of seconds a host or service check took to execute. This type of performance data is available for all checks that are performed. The *$HOSTEXECUTIONTIME$* and *$SERVICEEXECUTIONTIME$ macros* can be used to determine the number of seconds a host or service check was running and the *$HOSTLATENCY$* and *$SERVICE-LATENCY$* macros can be used to determine how "late" a regularly-scheduled host or service check was.

Plugin performance data is external data specific to the plugin used to perform the host or service check. Plugin-specific data can include things like percent packet loss, free disk space, processor load, number of current users, etc. - basically any type of metric that the plugin is measuring when it executes. Plugin-specific performance data is optional and may not be supported by all plugins. Plugin-specific performance data (if available) can be obtained by using the *$HOSTPERFDATA$* and *$SERVICEPERFDATA$ macros*. Read on for more information on how plugins can return performance data to Shinken for inclusion in the $HOSTPERFDATA$ and $SERVICEPERFDATA$ macros.

7.13.3 Plugin Performance Data

At a minimum, Shinken plugins must return a single line of human-readable text that indicates the status of some type of measurable data. For example, the **check_ping** plugin might return a line of text like the following:

```
PING ok - Packet loss = 0%, RTA = 0.80 ms
```

With this simple type of output, the entire line of text is available in the $HOSTOUTPUT$ or $SERVICEOUTPUT$ *macros* (depending on whether this plugin was used as a host check or service check).

Plugins can return optional performance data in their output by sending the normal, human-readable text string that they usually would, followed by a pipe character (|), and then a string containing one or more performance data metrics. Let's take the **check_ping** plugin as an example and assume that it has been enhanced to return percent packet loss and average round trip time as performance data metrics. Sample output from the plugin might look like this:

```
PING ok - Packet loss = 0%, RTA = 0.80 ms | percent_packet_loss=0, rta=0.80
```

When Shinken sees this plugin output format it will split the output into two parts:

- Everything before the pipe character is considered to be the "normal" plugin output and will be stored in either the $HOSTOUTPUT$ or $SERVICEOUTPUT$ macro
- Everything after the pipe character is considered to be the plugin-specific performance data and will be stored in the $HOSTPERFDATA$ or $SERVICEPERFDATA$ macro

In the example above, the $HOSTOUTPUT$ or $SERVICEOUTPUT$ macro would contain *"PING ok - Packet loss = 0%, RTA = 0.80 ms"* (without quotes) and the $HOSTPERFDATA$ or $SERVICEPERFDATA$ macro would contain *"percent_packet_loss=0, rta=0.80"* (without quotes).

Multiple lines of performace data (as well as normal text output) can be obtained from plugins, as described in the *plugin API documentation*.

The Shinken daemon doesn't directly process plugin performance data, so it doesn't really care what the performance data looks like. There aren't really any inherent limitations on the format or content of the performance data. However, if you are using an external addon to process the performance data (i.e. PerfParse), the addon may be expecting that the plugin returns performance data in a specific format. Check the documentation that comes with the addon for more information.

7.13.4 Processing Performance Data

If you want to process the performance data that is available from Shinken and the plugins, you'll need to do the following:

- Enable the *process_performance_data* option.
- Configure Shinken so that performance data is either written to files and/or processed by executing commands.

Read on for information on how to process performance data by writing to files or executing commands.

7.13.5 Processing Performance Data Using Commands

The most flexible way to process performance data is by having Shinken execute commands (that you specify) to process or redirect the data for later processing by external applications. The commands that Shinken executes to process host and service performance data are determined by the *host_perfdata_command* and *service_perfdata_command* options, respectively.

An example command definition that redirects service check performance data to a text file for later processing by another application is shown below:

```
define command{
  command_name    store-service-perfdata
  command_line    /bin/echo -e "$LASTSERVICECHECK$\t$HOSTNAME$\t$SERVICEDESC$\t$SERVICESTATE$\t$SERVI
}
```

This method, while flexible, comes with a relatively high CPU overhead. If you're processing performance data for a large number of hosts and services, you'll probably want Shinken to write performance data to files instead. This method is described in the next section.

7.13.6 Writing Performance Data To Files

You can have Shinken write all host and service performance data directly to text files using the *host_perfdata_file* and *service_perfdata_file* options. The format in which host and service performance data is written to those files is determined by the *host_perfdata_file_template* and service_perfdata_file_template options.

An example file format template for service performance data might look like this:

```
service_perfdata_file_template=[SERVICEPERFDATA]\t$TIMET$\t$HOSTNAME$\t$SERVICEDESC$\t$$ERVICEEXECUT
```

By default, the text files will be opened in "append" mode. If you need to change the modes to "write" or "non-blocking read/write" (useful when writing to pipes), you can use the host_perfdata_file_mode and service_perfdata_file_mode options.

Additionally, you can have Shinken periodically execute commands to periocially process the performance data files (e.g. rotate them) using the *host_perfdata_file_processing_command* and *service_perfdata_file_processing_command* options. The interval at which these commands are executed are governed by the *host_perfdata_file_processing_interval* and *service_perfdata_file_processing_interval* options, respectively.

7.14 Scheduled Downtime

7.14.1 Introduction

7.14.2 Scheduling Downtime

You can schedule downtime with your favorite UI or as an external command in cli.

Once you schedule downtime for a host or service, Shinken will add a comment to that host/service indicating that it is scheduled for downtime during the period of time you indicated. When that period of downtime passes, Shinken will automatically delete the comment that it added. Nice, huh?

7.14.3 Fixed vs. Flexible Downtime

When you schedule downtime for a host or service through the web interface you'll be asked if the downtime is fixed or flexible. Here's an explanation of how "fixed" and "flexible" downtime differs:

"Fixed" downtime starts and stops at the exact start and end times that you specify when you schedule it. Okay, that was easy enough...

"Flexible" downtime is intended for times when you know that a host or service is going to be down for X minutes (or hours), but you don't know exactly when that'll start. When you schedule flexible downtime, Shinken will start the scheduled downtime sometime between the start and end times you specified. The downtime will last for as long as the duration you specified when you scheduled the downtime. This assumes that the host or service for which you scheduled flexible downtime either goes down (or becomes unreachable) or goes into a non-OK state sometime between the start and end times you specified. The time at which a host or service transitions to a problem state determines the time at which Shinken actually starts the downtime. The downtime will then last for the duration you specified, even if the host or service recovers before the downtime expires. This is done for a very good reason. As we all know, you might think you've got a problem fixed, but then have to restart a server ten times before it actually works right. Smart, eh?

7.14.4 Triggered Downtime

When scheduling host or service downtime you have the option of making it "triggered" downtime. What is triggered downtime, you ask? With triggered downtime the start of the downtime is triggered by the start of some other scheduled host or service downtime. This is extremely useful if you're scheduling downtime for a large number or hosts or services and the start time of the downtime period depends on the start time of another downtime entry. For instance, if you schedule flexible downtime for a particular host (because its going down for maintenance), you might want to schedule triggered downtime for all of that hosts's "children".

7.14.5 How Scheduled Downtime Affects Notifications

When a host or service is in a period of scheduled downtime, Shinken will not allow normal notifications to be sent out for the host or service. However, a "DOWNTIMESTART" notification will get sent out for the host or service, which will serve to put any admins on notice that they won't receive upcoming problem alerts.

When the scheduled downtime is over, Shinken will allow normal notifications to be sent out for the host or service again. A "DOWNTIMEEND" notification will get sent out notifying admins that the scheduled downtime is over, and they will start receiving normal alerts again.

If the scheduled downtime is cancelled prematurely (before it expires), a "DOWNTIMECANCELLED" notification will get sent out to the appropriate admins.

7.14.6 Overlapping Scheduled Downtime

I like to refer to this as the "Oh crap, its not working" syndrome. You know what I'm talking about. You take a server down to perform a "routine" hardware upgrade, only to later realize that the OS drivers aren't working, the RAID array blew up, or the drive imaging failed and left your original disks useless to the world. Moral of the story is that any routine work on a server is quite likely to take three or four times as long as you had originally planned...

Let's take the following scenario:

- You schedule downtime for host A from 7:30pm-9:30pm on a Monday
- You bring the server down about 7:45pm Monday evening to start a hard drive upgrade
- After wasting an hour and a half battling with SCSI errors and driver incompatabilities, you finally get the machine to boot up
- At 9:15 you realize that one of your partitions is either hosed or doesn't seem to exist anywhere on the drive
- Knowing you're in for a long night, you go back and schedule additional downtime for host A from 9:20pm Monday evening to 1:30am Tuesday Morning.

If you schedule overlapping periods of downtime for a host or service (in this case the periods were 7:40pm-9:30pm and 9:20pm-1:30am), Shinken will wait until the last period of scheduled downtime is over before it allows notifications to be sent out for that host or service. In this example notifications would be suppressed for host A until 1:30am Tuesday morning.

7.15 Adaptive Monitoring

7.15.1 Introduction

Shinken allows you to change certain commands and host and service check attributes during runtime. I'll refer to this feature as "adaptive monitoring". Please note that the adaptive monitoring features found in Shinken will probably not be of much use to 99% of users, but they do allow you to do some neat things.

7.15.2 What Can Be Changed?

The following host/service check attributes can be changed during runtime:

- Check command (and command arguments)
- Check interval
- Max check attempts
- Check timeperiod
- Event handler command (and command arguments)

The following global attributes can be changed during runtime:

- Global host event handler command (and command arguments)
- Global service event handler command (and command arguments)

7.15.3 External Commands For Adaptive Monitoring

In order to change global or host- or service-specific attributes during runtime, you must submit the appropriate *external command* to Shinken. The table below lists the different attributes that may be changed during runtime, along with the external command to accomplish the job.

A full listing of external commands that can be used for adaptive monitoring (along with examples of how to use them) can be found online at the following URL: http://www.nagios.org/developerinfo/externalcommands/

- When changing check commands, check timeperiods, or event handler commands, it is important to note that the new values for these options must have been defined before shinken was started. Any request to change a command or timeperiod to one which had not been defined when it was started is ignored.
- You can specify command arguments along with the actual command name - just seperate individual arguments from the command name (and from each other) using bang (!) characters. More information on how arguments in command definitions are processed during runtime can be found in the documentation on *macros*.

7.16 Predictive Dependency Checks

Note: The predictive dependency check functionality is not managed from now in Shinken.

7.16.1 Introduction

Host and service *dependencies* can be defined to allow you greater control over when checks are executed and when notifications are sent out. As dependencies are used to control basic aspects of the monitoring process, it is crucial to ensure that status information used in the dependency logic is as up to date as possible.

Shinken allows you to enable predictive dependency checks for hosts and services to ensure that the dependency logic will have the most up-to-date status information when it comes to making decisions about whether to send out notifications or allow active checks of a host or service.

7.16.2 How Do Predictive Checks Work?

The image below shows a basic diagram of hosts that are being monitored by Shinken, along with their parent/child relationships and dependencies.

The *Switch2* host in this example has just changed state from an UP state to a problem state. Shinken needs to determine whether the host is DOWN or UNREACHABLE, so it will launch parallel checks of *Switch2*'s immediate parents (*Firewall1*) and children (*Comp1*, *Comp2*, and *Switch3*). This is a normal function of the *host reachability* logic.

You will also notice that *Switch2* is depending on *Monitor1* and *File1* for either notifications or check execution (which one is unimportant in this example). If predictive host dependency checks are enabled, Shinken will launch parallel checks of Monitor1 and File1 at the same time it launches checks of *Switch2*'s immediate parents and children. Shinken does this because it knows that it will have to test the dependency logic in the near future (e.g. for purposes of notification) and it wants to make sure it has the most current status information for the hosts that take part in the dependency.

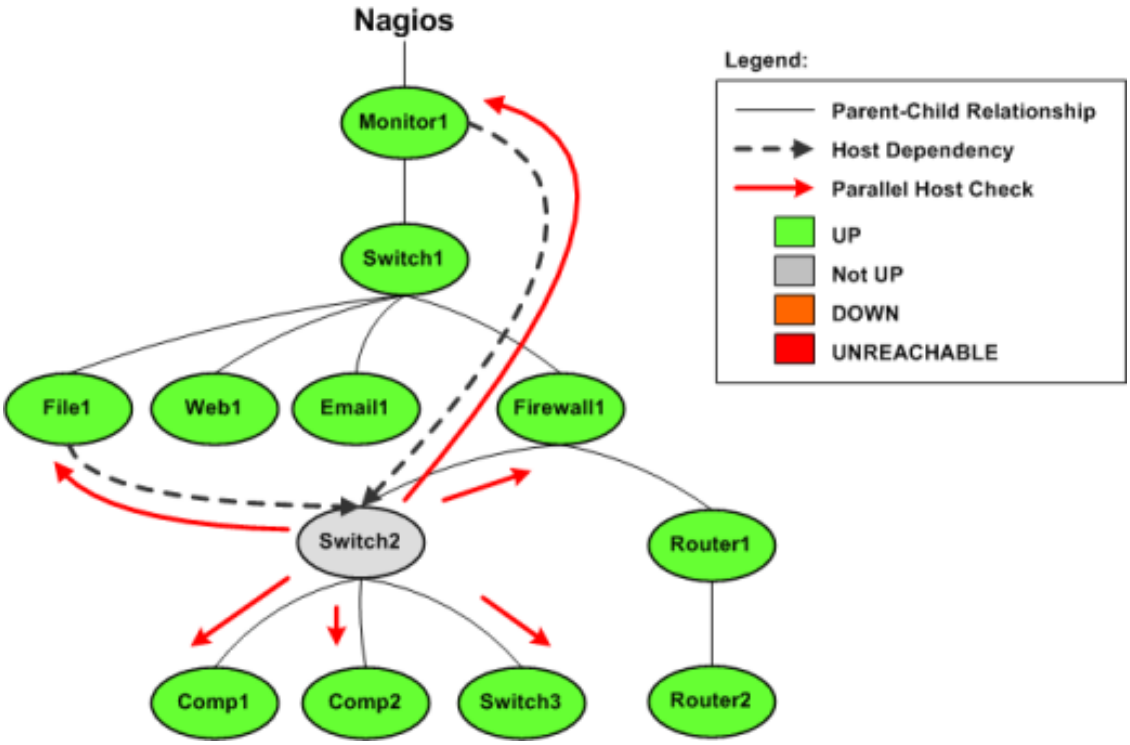

That's how predictive dependency checks work. Simple, eh?

Predictive service dependency checks work in a similiar manner to what is described above. Except, of course, they deal with services instead of hosts.

7.16.3 Enabling Predictive Checks

Predictive dependency checks involve rather little overhead, so I would recommend that you enable them. In most cases, the benefits of having accurate information for the dependency logic outweighs the extra overhead imposed by these checks.

Enabling predictive dependency checks is easy:

- Predictive host dependency checks are controlled by the *"enable_predictive_host_dependency_checks"* option.

- Predictive service dependency checks are controlled by the *"enable_predictive_service_dependency_checks"* option.

7.16.4 Cached Checks

Predictive dependency checks are on-demand checks and are therefore subject to the rules of *cached checks*. Cached checks can provide you with performance improvements by allowing Shinken to forgo running an actual host or service check if it can use a relatively recent check result instead. More information on cached checks can be found *here*.

7.17 Cached Checks

7.17.1 Introduction

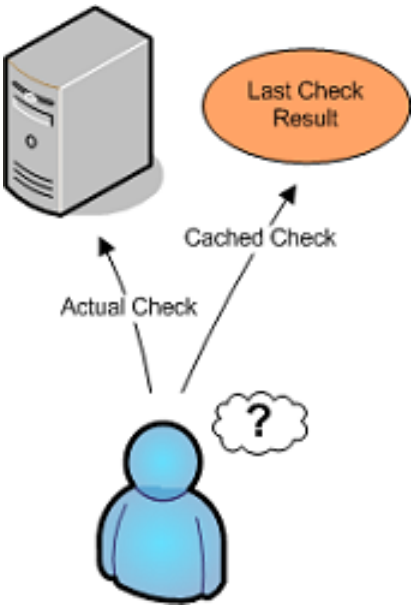

The performance of Shinken' monitoring logic can be significantly improved by implementing the use of cached checks. Cached checks allow Shinken to forgot executing a host or service check command if it determines a relatively recent check result will do instead.

7.17.2 For On-Demand Checks Only

Regularly scheduled host and service checks will not see a performance improvement with use of cached checks. Cached checks are only useful for improving the performance of on-demand host and service checks. Scheduled checks help to ensure that host and service states are updated regularly, which may result in a greater possibility their results can be used as cached checks in the future.

For reference, on-demand host checks occur...

- When a service associated with the host changes state.
- As needed as part of the *host reachability* logic.
- As needed for *host dependency checks*.

And on-demand service checks occur...

- As needed for *service dependency checks*.

Unless you make use of service dependencies, Shinken will not be able to use cached check results to improve the performance of service checks. Don't worry about that - its normal. Cached host checks are where the big performance improvements lie, and everyone should see a benefit there.

7.17.3 How Caching Works

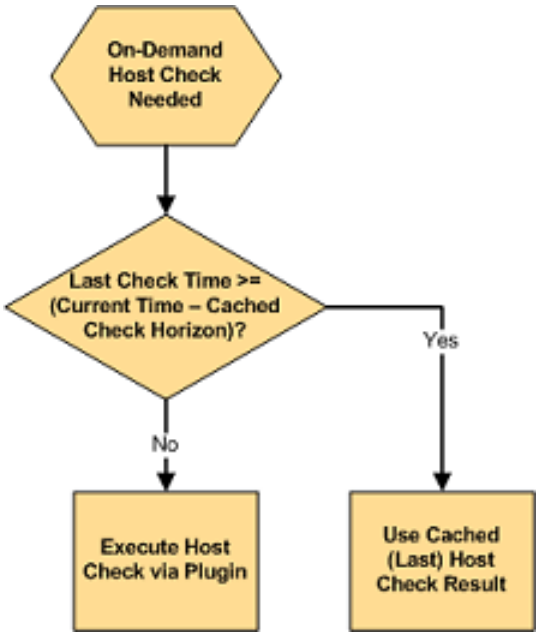

When Shinken needs to perform an on-demand host or service check, it will make a determination as to whether it can used a cached check result or if it needs to perform an actual check by executing a plugin. It does this by checking to see if the last check of the host or service occured within the last X seconds, where X is the cached host or service check horizon.

If the last check was performed within the timeframe specified by the cached check horizon variable, Shinken will use the result of the last host or service check and will not execute a new check. If the host or service has not yet been checked, or if the last check falls outside of the cached check horizon timeframe, Shinken will execute a new host or service check by running a plugin.

7.17.4 What This Really Means

Shinken performs on-demand checks because it need to know the current state of a host or service at that exact moment in time. Utilizing cached checks allows you to make Shinken think that recent check results are "good enough" for determining the current state of hosts, and that it doesn't need to go out and actually re-check the status of that host or service.

The cached check horizon tells Shinken how recent check results must be in order to reliably reflect the current state of a host or service. For example, with a cached check horizon of 30 seconds, you are telling Shinken that if a host's state was checked sometime in the last 30 seconds, the result of that check should still be considered the current state of the host.

The number of cached check results that Shinken can use versus the number of on-demand checks it has to actually execute can be considered the cached check "hit" rate. By increasing the cached check horizon to equal the regular check interval of a host, you could theoretically achieve a cache hit rate of 100%. In that case all on-demand checks of that host would use cached check results. What a performance improvement! But is it really? Probably not.

The reliability of cached check result information decreases over time. Higher cache hit rates require that previous check results are considered "valid" for longer periods of time. Things can change quickly in any network scenario, and there's no guarantee that a server that was functioning properly 30 seconds ago isn't on fire right now. There's the tradeoff - reliability versus speed. If you have a large cached check horizon, you risk having unreliable check result values being used in the monitoring logic.

Shinken will eventually determine the correct state of all hosts and services, so even if cached check results prove to unreliably represent their true value, it will only work with incorrect information for a short period of time. Even short periods of unreliable status information can prove to be a nuisance for admins, as they may receive notifications about problems which no longer exist.

There is no standard cached check horizon or cache hit rate that will be acceptable to every users. Some people will want a short horizon timeframe and a low cache hit rate, while others will want a larger horizon timeframe and a larger cache hit rate (with a low reliability rate). Some users may even want to disable cached checks altogether to obtain a 100% reliability rate. Testing different horizon timeframes, and their effect on the reliability of status information, is the only want that an individual user will find the "right" value for their situation. More information on this is discussed below.

7.17.5 Configuration Variables

The following variables determine the timeframes in which a previous host or service check result may be used as a cached host or service check result:

- The *cached_host_check_horizon* variable controls cached host checks.
- The *cached_service_check_horizon* variable controls cached service checks.

7.17.6 Optimizing Cache Effectiveness

In order to make the most effective use of cached checks, you should:

- Schedule regular checks of your hosts
- Use MRTG to graph statistics for 1) on-demand checks and 2) cached checks
- Adjust cached check horizon variables to fit your needs

You can schedule regular checks of your hosts by specifying a value greater than 0 for check_interval option in your *host definitions*.

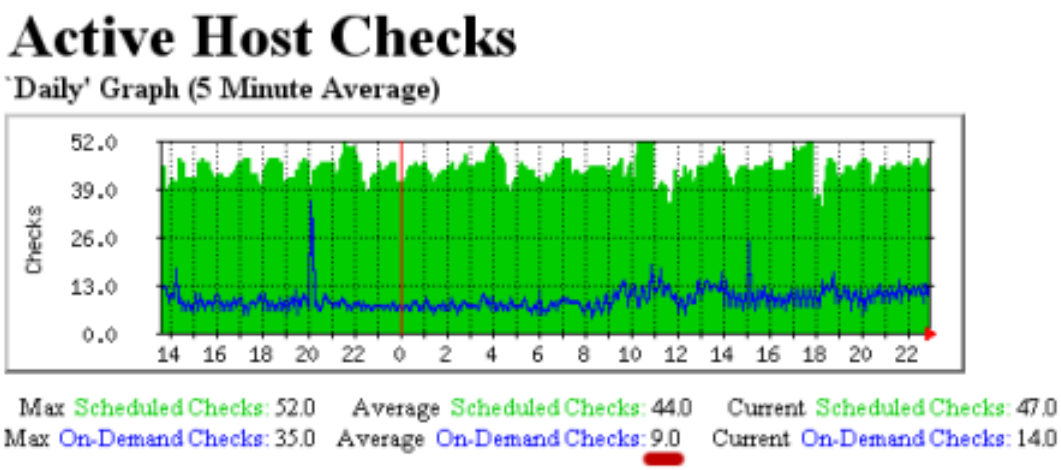

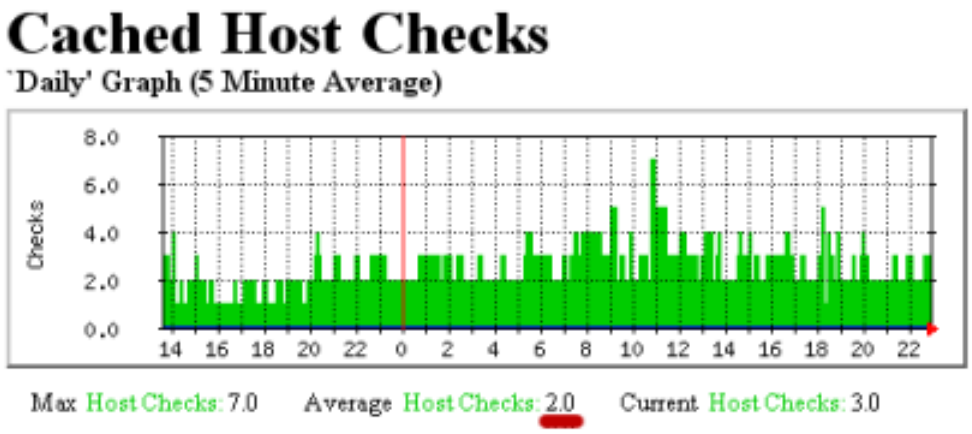

A good way to determine the proper value for the cached check horizon options is to compare how many on-demand checks Shinken has to actually run versus how may it can use cached values for. The nagiostats utility can produce information on cached checks, which can then be graphed with MRTG. Example MRTG graphs that show cached vs. actual on-demand checks are shown to the right.

The monitoring installation which produced the graphs above had:

- A total of 44 hosts, all of which were checked at regular intervals
- An average (regularly scheduled) host check interval of 5 minutes
- A *cached_host_check_horizon* of 15 seconds

The first MRTG graph shows how many regularly scheduled host checks compared to how many cached host checks have occured. In this example, an average of 53 host checks occur every five minutes. 9 of these (17%) are on-demand checks.

The second MRTG graph shows how many cached host checks have occurred over time. In this example an average of 2 cached host checks occurs every five minutes.

Remember, cached checks are only available for on-demand checks. Based on the 5 minute averages from the graphs, we see that Nagios is able to used cached host check results every 2 out of 9 times an on-demand check has to be run. That may not seem much, but these graphs represent a small monitoring environment. Consider that 2 out of 9 is 22% and you can start to see how this could significantly help improve host check performance in large environments. That percentage could be higher if the cached host check horizon variable value was increased, but that would reduce the reliability of the cached host state information.

Once you've had a few hours or days worth of MRTG graphs, you should see how many host and service checks were done by executing plugins versus those that used cached check results. Use that information to adjust the cached check horizon variables appropriately for your situation. Continue to monitor the MRTG graphs over time to see how changing the horizon variables affected cached check statistics. Rinse and repeat as necessary.

7.18 Passive Host State Translation

7.18.1 Introduction

This Nagios option is no more useful in the Shinken architecture.

7.19 Service and Host Check Scheduling

7.19.1 The scheduling

The scheduling of Shinken is quite simple. The first scheduling take care of the max_service_check_spread and max_host_check_spread so the time of the first schedule will be in the

```
start+max_*_check_spread*interval_length (60s in general)
```

if the check_timeperiod agree with it.

Note: Shinken do not take care about Nagios *_inter_check_delay_method : this is always 's' (smart) because other options are just useless for nearly everyone. And it also do not use the *_interleave_factor too.

Nagios make a average of service by host to make it's dispatch of checks in the first check window. Shinken use a random way of doing it : the check is between t=now and t=min(t from next timeperiod, max_*_check_spread), but in a random way. So you will will have the better distribution of checks in this period, intead of the nagios one where hosts with differents number of services can be agresively checks.

After this first scheduling, the time for the next check is just t_check+check_interval if the timepriod is agree for it (or just the next time available in the timeperiod). In the future, a little random value (like few seconds) will be add for such cases.

7.20 Object Inheritance

7.20.1 Introduction

This documentation attempts to explain object inheritance and how it can be used in your *object definitions*.

If you are confused about how recursion and inheritance work after reading this, take a look at the sample object config files provided in the Shinken distribution. If that still doesn't help, have a look to the *shinken resources for help*.

7.20.2 Basics

There are three variables affecting recursion and inheritance that are present in all object definitions. They are indicated in red as follows...

```
define someobjecttype{
                      object-specific variables ...
                      name            template_name
                      use             name_of_template_to_use
                      register        [0/1]
                      }
```

The first variable is "name". Its just a "template" name that can be referenced in other object definitions so they can inherit the objects properties/variables. Template names must be unique amongst objects of the same type, so you can't have two or more host definitions that have "hosttemplate" as their template name.

The second variable is "use". This is where you specify the name of the template object that you want to inherit properties/variables from. The name you specify for this variable must be defined as another object's template named (using the name variable).

The third variable is "register". This variable is used to indicate whether or not the object definition should be "registered" with Shinken. By default, all object definitions are registered. If you are using a partial object definition as a template, you would want to prevent it from being registered (an example of this is provided later). Values are as follows: 0 = do NOT register object definition, 1 = register object definition (this is the default). This variable is NOT inherited; every (partial) object definition used as a template must explicitly set the "register" directive to be 0. This prevents the need to override an inherited "register" directive with a value of 1 for every object that should be registered.

7.20.3 Local Variables vs. Inherited Variables

One important thing to understand with inheritance is that "local" object variables always take precedence over variables defined in the template object. Take a look at the following example of two host definitions (not all required variables have been supplied):

```
define host{
                       host_name               bighost1
                       check_command           check-host-alive
                       notification_options    d,u,r
                       max_check_attempts      5
                       name                    hosttemplate1
                       }

                define host{
                       host_name               bighost2
                       max_check_attempts      3
                       use                     hosttemplate1
                       }
```

You'll note that the definition for host *bighost1* has been defined as having *hosttemplate1* as its template name. The definition for host *bighost2* is using the definition of *bighost1* as its template object. Once Shinken processes this data, the resulting definition of host *bighost2* would be equivalent to this definition:

```
define host{
                      host_name               bighost2
                      check_command           check-host-alive
                      notification_options    d,u,r
                      max_check_attempts      3
                      }
```

You can see that the "check_command" and "notification_options" variables were inherited from the template object (where host *bighost1* was defined). However, the *host_name* and *max_check_attempts* variables were not inherited

from the template object because they were defined locally. Remember, locally defined variables override variables that would normally be inherited from a template object. That should be a fairly easy concept to understand.

If you would like local string variables to be appended to inherited string values, you can do so. Read more about how to accomplish this *below*.

7.20.4 Inheritance Chaining

Objects can inherit properties/variables from multiple levels of template objects. Take the following example:

```
define host{
                        host_name               bighost1
                        check_command           check-host-alive
                        notification_options    d,u,r
                        max_check_attempts      5
                        name                    hosttemplate1
                        }

                define host{
                        host_name               bighost2
                        max_check_attempts      3
                        use                     hosttemplate1
                        name                    hosttemplate2
                        }

                define host{
                        host_name               bighost3
                        use                     hosttemplate2
                        }
```

You'll notice that the definition of host *bighost3* inherits variables from the definition of host *bighost2*, which in turn inherits variables from the definition of host *bighost1*. Once Shinken processes this configuration data, the resulting host definitions are equivalent to the following:

```
define host{
                        host_name               bighost1
                        check_command           check-host-alive
                        notification_options    d,u,r
                        max_check_attempts      5
                        }

                define host{
                        host_name               bighost2
                        check_command           check-host-alive
                        notification_options    d,u,r
                        max_check_attempts      3
                        }

                define host{
                        host_name               bighost3
                        check_command           check-host-alive
                        notification_options    d,u,r
                        max_check_attempts      3
                        }
```

There is no inherent limit on how "deep" inheritance can go, but you'll probably want to limit yourself to at most a few levels in order to maintain sanity.

7.20.5 Using Incomplete Object Definitions as Templates

It is possible to use imcomplete object definitions as templates for use by other object definitions. By "incomplete" definition, I mean that all required variables in the object have not been supplied in the object definition. It may sound odd to use incomplete definitions as templates, but it is in fact recommended that you use them. Why? Well, they can serve as a set of defaults for use in all other object definitions. Take the following example:

```
define host{
                        check_command           check-host-alive
                        notification_options    d,u,r
                        max_check_attempts      5
                        name                    generichosttemplate
                        register                        0
                        }

                define host{
                        host_name               bighost1
                        address                 192.168.1.3
                        use                     generichosthosttemplate
                        }

                define host{
                        host_name               bighost2
                        address                 192.168.1.4
                        use                     generichosthosttemplate
                        }
```

Notice that the first host definition is incomplete because it is missing the required "host_name" variable. We don't need to supply a host name because we just want to use this definition as a generic host template. In order to prevent this definition from being registered with Shinken as a normal host, we set the "register" variable to 0.

The definitions of hosts *bighost1* and *bighost2* inherit their values from the generic host definition. The only variable we've chosed to override is the "address" variable. This means that both hosts will have the exact same properties, except for their "host_name" and "address" variables. Once Shinken processes the config data in the example, the resulting host definitions would be equivalent to specifying the following:

```
define host{
                        host_name               bighost1
                        address                 192.168.1.3
                        check_command           check-host-alive
                        notification_options    d,u,r
                        max_check_attempts      5
                        }

                define host{
                        host_name               bighost2
                        address                 192.168.1.4
                        check_command           check-host-alive
                        notification_options    d,u,r
                        max_check_attempts      5
                        }
```

At the very least, using a template definition for default variables will save you a lot of typing. It'll also save you a lot of headaches later if you want to change the default values of variables for a large number of hosts.

7.20.6 Custom Object Variables

Any *custom object variables* that you define in your host, service, or contact definition templates will be inherited just like other standard variables. Take the following example:

```
define host{
                        _customvar1             somevalue  ; <-- Custom host variable
                        _snmp_community         public  ; <-- Custom host variable
                        name                    generichosttemplate
                        register                        0
                        }

                define host{
                        host_name               bighost1
                        address                 192.168.1.3
                        use                     generichosthosttemplate
                        }
```

The host *bighost1* will inherit the custom host variables "_customvar1" and "_snmp_community", as well as their respective values, from the *generichosttemplate* definition. The effective result is a definition for *bighost1* that looks like this:

```
define host{
                      host_name               bighost1
                      address                 192.168.1.3
                      _customvar1             somevalue
                      _snmp_community         public
                      }
```

7.20.7 Cancelling Inheritance of String Values

In some cases you may not want your host, service, or contact definitions to inherit values of string variables from the templates they reference. If this is the case, you can specify **"null"** (without quotes) as the value of the variable that you do not want to inherit. Take the following example:

```
define host{
                       event_handler           my-event-handler-command
                       name                    generichosttemplate
                       register                        0
                       }

                define host{
                       host_name               bighost1
                       address                 192.168.1.3
                       event_handler   null
                       use                     generichosthosttemplate
                       }
```

In this case, the host *bighost1* will not inherit the value of the "event_handler" variable that is defined in the *generichosttemplate*. The resulting effective definition of *bighost1* is the following:

```
define host{
                       host_name               bighost1
                       address                 192.168.1.3
                       }
```

7.20.8 Additive Inheritance of String Values

Shinken gives preference to local variables instead of values inherited from templates. In most cases local variable values override those that are defined in templates. In some cases it makes sense to allow Shinken to use the values of inherited and local variables together.

This "additive inheritance" can be accomplished by prepending the local variable value with a plus sign (+). This features is only available for standard (non-custom) variables that contain string values. Take the following example:

```
define host{
                      hostgroups              all-servers
                      name                    generichosttemplate
                      register                        0
                      }

              define host{
                      host_name                       linuxserver1
                      hostgroups            +linux-servers,web-servers
                      use                    generichosthosttemplate
                      }
```

In this case, the host *linuxserver1* will append the value of its local "hostgroups" variable to that from generichosttemplate. The resulting effective definition of *linuxserver1* is the following:

```
define host{
                      host_name                       linuxserver1
                      hostgroups        all-servers,linux-servers,web-servers
                      }
```

Important: If you use a field twice using several templates, the value of the field will be the first one found! In the example above, fields values in all-servers won't we be replaced. Be careful with overlaping field!

7.20.9 Implied Inheritance

Normally you have to either explicitly specify the value of a required variable in an object definition or inherit it from a template. There are a few exceptions to this rule, where Shinken will assume that you want to use a value that instead comes from a related object. For example, the values of some service variables will be copied from the host the service is associated with if you don't otherwise specify them.

The following table lists the object variables that will be implicitly inherited from related objects if you don't explicitly specify their value in your object definition or inherit them from a template.

Object Type	Object Variable	Implied Source
Services	*contact_groups*	*contact_groups* in the associated host definition
notification_interval	*notification_interval* in the associated host definition	
notification_period	*notification_period* in the associated host definition	
check_period	*check_period* in the associated host definition	
Host Escalations	*contact_groups*	*contact_groups* in the associated host definition
notification_interval	*notification_interval* in the associated host definition	
escalation_period	*notification_period* in the associated host definition	
Service Escalations	*contact_groups*	*contact_groups* in the associated service definition
notification_interval	*notification_interval* in the associated service definition	
escalation_period	*notification_period* in the associated service definition	

7.20.10 Implied/Additive Inheritance in Escalations

Service and host escalation definitions can make use of a special rule that combines the features of implied and additive inheritance. If escalations 1) do not inherit the values of their "contact_groups" or "contacts" directives from another escalation template and 2) their "contact_groups" or "contacts" directives begin with a plus sign (+), then the values of their corresponding host or service definition's "contact_groups" or "contacts" directives will be used in the additive inheritance logic.

Confused? Here's an example:

```
define host{
              name            linux-server
              contact_groups  linux-admins
              ...
              }

      define hostescalation{
              host_name               linux-server
              contact_groups  +management
              ...
              }
```

This is a much simpler equivalent to:

```
define hostescalation{
              host_name               linux-server
              contact_groups  linux-admins,management
              ...
              }
```

7.20.11 Multiple Inheritance Sources

Thus far, all examples of inheritance have shown object definitions inheriting variables/values from just a single source. You are also able to inherit variables/values from multiple sources for more complex configurations, as shown below.

```
# Generic host template

define host{
  name                        generic-host
  active_checks_enabled       1
  check_interval              10
  register                    0
}
```

```
# Development web server template
define host{
 name                    development-server
 check_interval          15
 notification_options    d,u,r
 ...
 register                        0
}
```

```
# Development web server
define host{
  use                    generic-host,development-server
  host_name              devweb1
  ...
}
```

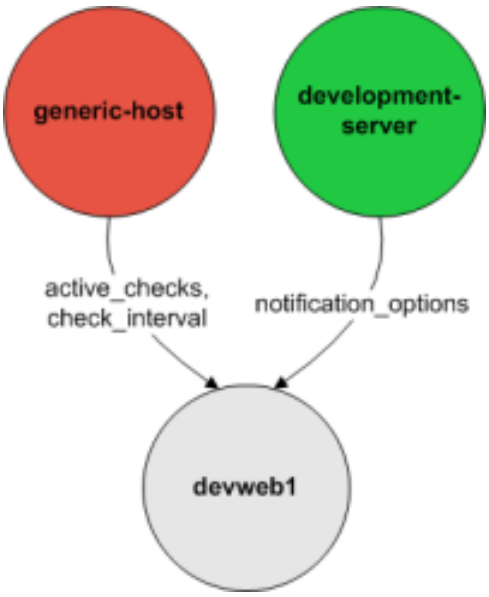

In the example above, devweb1 is inheriting variables/values from two sources: generic-host and development-server. You'll notice that a check_interval variable is defined in both sources. Since generic-host was the first template specified in devweb1's use directive, its value for the "check_interval" variable is inherited by the devweb1 host. After inheritance, the effective definition of devweb1 would be as follows:

```
# Development web serve
define host{
              host_name               devweb1
              active_checks_enabled   1
              check_interval          10
              notification_options    d,u,r
              ...
}
```

7.20.12 Precedence With Multiple Inheritance Sources

When you use multiple inheritance sources, it is important to know how Shinken handles variables that are defined in multiple sources. In these cases Shinken will use the variable/value from the first source that is specified in the use directive. Since inheritance sources can themselves inherit variables/values from one or more other sources, it can get tricky to figure out what variable/value pairs take precedence.

Consider the following host definition that references three templates:

```
# Development web server
define host{
           use        1, 4, 8
           host_name  devweb1
               ...
}
```

If some of those referenced templates themselves inherit variables/values from one or more other templates, the precendence rules are shown below. Testing, trial, and error will help you better understand exactly how things work in complex inheritance situations like this. :-)

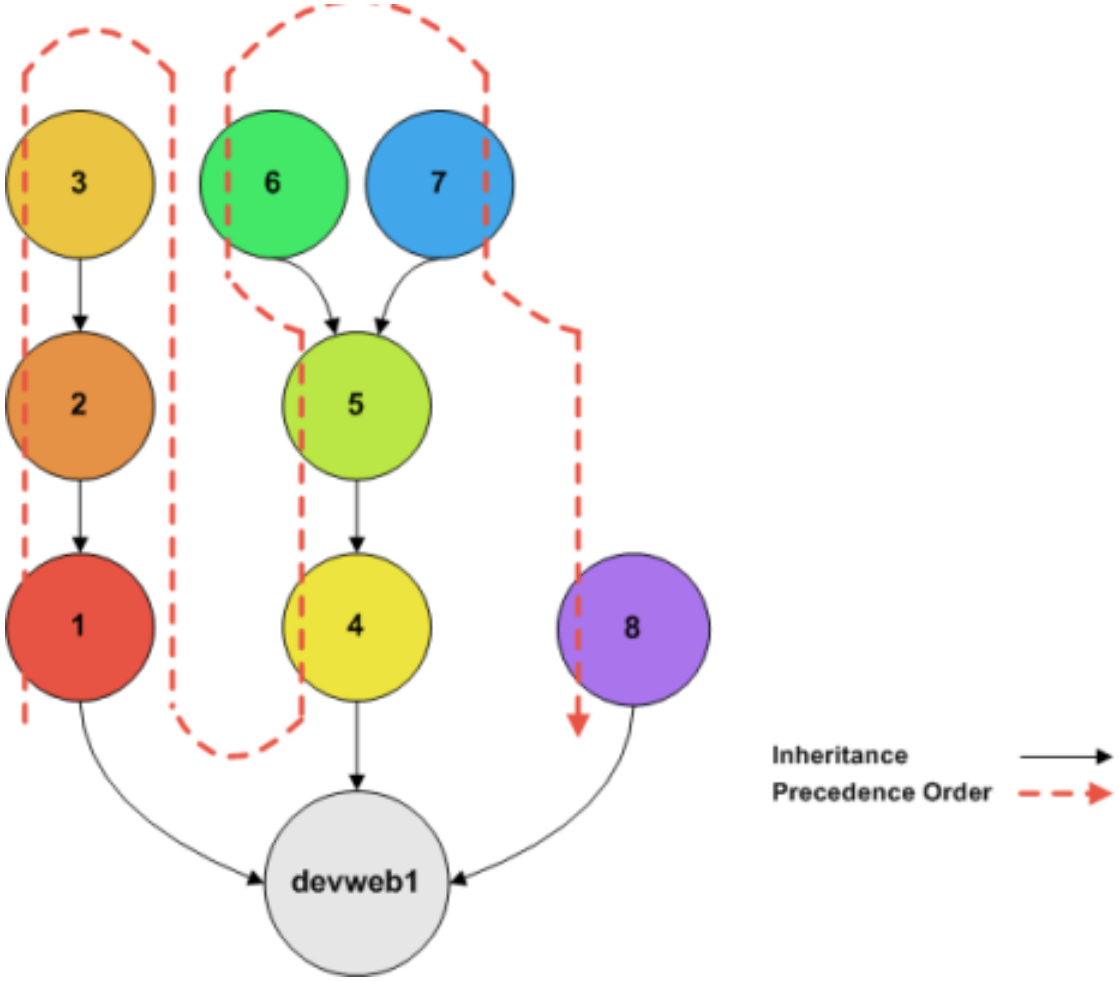

7.21 Advanced tricks

7.21.1 Time-Saving Tricks For Object Definitions

Abstract

or..."How To Preserve Your Sanity"

7.21.2 Introduction

This documentation attempts to explain how you can exploit the (somewhat) hidden features of *template-based object definitions* to save your sanity. How so, you ask? Several types of objects allow you to specify multiple host names and/or hostgroup names in definitions, allowing you to "copy" the object definition to multiple hosts or services. I'll cover each type of object that supports these features separately. For starters, the object types which support this time-saving feature are as follows:

- *Services*
- *Service escalations*
- *Service dependencies*
- *Host escalations*
- *Host dependencies*
- *Hostgroups*

Object types that are not listed above (i.e. timeperiods, commands, etc.) do not support the features I'm about to describe.

7.21.3 Service Definitions

Multiple Hosts:

If you want to create identical *services* that are assigned to multiple hosts, you can specify multiple hosts in the "host_name" directive. The definition below would create a service called SOMESERVICE on hosts HOST1 through HOSTN. All the instances of the SOMESERVICE service would be identical (i.e. have the same check command, max check attempts, notification period, etc.).

```
define service{
    host_name                HOST1,HOST2,HOST3,...,HOSTN
    service_description      SOMESERVICE
    other service directives ...
}
```

All Hosts In Multiple Hostgroups:

If you want to create identical services that are assigned to all hosts in one or more hostgroups, you can do so by creating a single service definition. How ? The "hostgroup_name" directive allows you to specify the name of one or more hostgroups that the service should be created for. The definition below would create a service called SOMESERVICE on all hosts that are members of hostgroups HOSTGROUP1 through HOSTGROUPN. All the instances of the SOMESERVICE service would be identical (i.e. have the same check command, max check attempts, notification period, etc.).

```
define service{
    hostgroup_name          HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
    service_description     SOMESERVICE
    other service directives ...
}
```

All Hosts:

If you want to create identical services that are assigned to all hosts that are defined in your configuration files, you can use a wildcard in the "host_name" directive. The definition below would create a service called SOMESERVICE on all hosts that are defined in your configuration files. All the instances of the SOMESERVICE service would be identical (i.e. have the same check command, max check attempts, notification period, etc.).

```
define service{
    host_name               *
    service_description     SOMESERVICE
    other service directives ...
}
```

Excluding Hosts:

If you want to create identical services on numerous hosts or hostgroups, but would like to exclude some hosts from the definition, this can be accomplished by preceding the host or hostgroup with a ! symbol.

```
define service{
    host_name             HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
    hostgroup_name        HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
    service_description   SOMESERVICE
    other service directives ...
}
```

7.21.4 Service Escalation Definitions

Multiple Hosts:

If you want to create *service escalations* for services of the same name/description that are assigned to multiple hosts, you can specify multiple hosts in the "host_name" directive. The definition below would create a service escalation for services called SOMESERVICE on hosts HOST1 through HOSTN. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define serviceescalation{
    host_name               HOST1,HOST2,HOST3,...,HOSTN
    service_description     SOMESERVICE
    other escalation directives ...
}
```

All Hosts In Multiple Hostgroups:

If you want to create service escalations for services of the same name/description that are assigned to all hosts in in one or more hostgroups, you can do use the "hostgroup_name" directive. The definition below would create a service escalation for services called SOMESERVICE on all hosts that are members of hostgroups HOSTGROUP1 through

HOSTGROUPN. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define serviceescalation{
    hostgroup_name          HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
    service_description     SOMESERVICE
    other escalation directives ...
}
```

All Hosts:

If you want to create identical service escalations for services of the same name/description that are assigned to all hosts that are defined in your configuration files, you can use a wildcard in the "host_name" directive. The definition below would create a service escalation for all services called SOMESERVICE on all hosts that are defined in your configuration files. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define serviceescalation{
    host_name               *
    service_description     SOMESERVICE
    other escalation directives ...
}
```

Excluding Hosts:

If you want to create identical services escalations for services on numerous hosts or hostgroups, but would like to exclude some hosts from the definition, this can be accomplished by preceding the host or hostgroup with a ! symbol.

```
define serviceescalation{
    host_name             HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
    hostgroup_name        HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
    service_description   SOMESERVICE
    other escalation directives ...
}
```

All Services On Same Host:

If you want to create *service escalations* for all services assigned to a particular host, you can use a wildcard in the "service_description" directive. The definition below would create a service escalation for all services on host HOST1. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

If you feel like being particularly adventurous, you can specify a wildcard in both the "host_name" and "service_description" directives. Doing so would create a service escalation for all services that you've defined in your configuration files.

```
define serviceescalation{
    host_name               HOST1
    service_description     *
    other escalation directives ...
}
```

Multiple Services On Same Host:

If you want to create *service escalations* for all multiple services assigned to a particular host, you can use a specify more than one service description in the "service_description" directive. The definition below would create a service escalation for services SERVICE1 through SERVICEN on host HOST1. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define serviceescalation{
    host_name               HOST1
    service_description     SERVICE1,SERVICE2,...,SERVICEN
    other escalation directives ...
}
```

All Services In Multiple Servicegroups:

If you want to create service escalations for all services that belong in one or more servicegroups, you can do use the "servicegroup_name" directive. The definition below would create service escalations for all services that are members of servicegroups SERVICEGROUP1 through SERVICEGROUPN. All the instances of the service escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define serviceescalation{
    servicegroup_name          SERVICEGROUP1,SERVICEGROUP2,...,SERVICEGROUPN
    other escalation directives ...
}
```

7.21.5 Service Dependency Definitions

Multiple Hosts:

If you want to create *service dependencies* for services of the same name/description that are assigned to multiple hosts, you can specify multiple hosts in the "host_name" and or "dependent_host_name" directives. In the example below, service SERVICE2 on hosts HOST3 and HOST4 would be dependent on service SERVICE1 on hosts HOST1 and HOST2. All the instances of the service dependencies would be identical except for the host names (i.e. have the same notification failure criteria, etc.).

```
define servicedependency{
    host_name                       HOST1,HOST2
    service_description             SERVICE1
    dependent_host_name             HOST3,HOST4
    dependent_service_description   SERVICE2
    other dependency directives ...
}
```

All Hosts In Multiple Hostgroups:

If you want to create service dependencies for services of the same name/description that are assigned to all hosts in in one or more hostgroups, you can do use the "hostgroup_name" and/or "dependent_hostgroup_name" directives. In the example below, service SERVICE2 on all hosts in hostgroups HOSTGROUP3 and HOSTGROUP4 would be dependent on service SERVICE1 on all hosts in hostgroups HOSTGROUP1 and HOSTGROUP2. Assuming there were five hosts in each of the hostgroups, this definition would be equivalent to creating 100 single service dependency definitions ! All the instances of the service dependency would be identical except for the host names (i.e. have the same notification failure criteria, etc.).

```
define servicedependency{
    hostgroup_name                  HOSTGROUP1,HOSTGROUP2
    service_description             SERVICE1
    dependent_hostgroup_name        HOSTGROUP3,HOSTGROUP4
    dependent_service_description   SERVICE2
    other dependency directives ...
}
```

All Services On A Host:

If you want to create service dependencies for all services assigned to a particular host, you can use a wildcard in the "service_description" and/or "dependent_service_description" directives. In the example below, all services on host HOST2 would be dependent on all services on host HOST1. All the instances of the service dependencies would be identical (i.e. have the same notification failure criteria, etc.).

```
define servicedependency{
    host_name                       HOST1
    service_description             *
    dependent_host_name             HOST2
    dependent_service_description   *
    other dependency directives ...
}
```

Multiple Services On A Host:

If you want to create service dependencies for multiple services assigned to a particular host, you can specify more than one service description in the "service_description" and/or "dependent_service_description" directives as follows:

```
define servicedependency{
    host_name                       HOST1
    service_description             SERVICE1,SERVICE2,...,SERVICEN
    dependent_host_name             HOST2
    dependent_service_description   SERVICE1,SERVICE2,...,SERVICEN
    other dependency directives ...
}
```

All Services In Multiple Servicegroups:

If you want to create service dependencies for all services that belong in one or more servicegroups, you can do use the "servicegroup_name" and/or "dependent_servicegroup_name" directive as follows:

```
define servicedependency{
    servicegroup_name               SERVICEGROUP1,SERVICEGROUP2,...,SERVICEGROUPN
    dependent_servicegroup_name     SERVICEGROUP3,SERVICEGROUP4,...SERVICEGROUPN
    other dependency directives ...
}
```

Same Host Dependencies:

If you want to create service dependencies for multiple services that are dependent on services on the same host, leave the "dependent_host_name" and "dependent_hostgroup_name" directives empty. The example below assumes that hosts HOST1 and HOST2 have at least the following four services associated with them: SERVICE1, SERVICE2, SERVICE3, and SERVICE4. In this example, SERVICE3 and SERVICE4 on HOST1 will be dependent on both

SERVICE1 and SERVICE2 on HOST1. Similiarly, SERVICE3 and SERVICE4 on HOST2 will be dependent on both SERVICE1 and SERVICE2 on HOST2.

```
define servicedependency{
    host_name                       HOST1,HOST2
    service_description             SERVICE1,SERVICE2
    dependent_service_description   SERVICE3,SERVICE4
    other dependency directives ...
}
```

7.21.6 Host Escalation Definitions

Multiple Hosts:

If you want to create *host escalations* for multiple hosts, you can specify multiple hosts in the "host_name" directive. The definition below would create a host escalation for hosts HOST1 through HOSTN. All the instances of the host escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define hostescalation{
    host_name              HOST1,HOST2,HOST3,...,HOSTN
    other escalation directives ...
}
```

All Hosts In Multiple Hostgroups:

If you want to create host escalations for all hosts in in one or more hostgroups, you can do use the "hostgroup_name" directive. The definition below would create a host escalation on all hosts that are members of hostgroups HOSTGROUP1 through HOSTGROUPN. All the instances of the host escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define hostescalation{
    hostgroup_name            HOSTGROUP1,HOSTGROUP2,...,HOSTGROUPN
    other escalation directives ...
}
```

All Hosts:

If you want to create identical host escalations for all hosts that are defined in your configuration files, you can use a wildcard in the "host_name" directive. The definition below would create a hosts escalation for all hosts that are defined in your configuration files. All the instances of the host escalation would be identical (i.e. have the same contact groups, notification interval, etc.).

```
define hostescalation{
    host_name               *
    other escalation directives ...
}
```

Excluding Hosts:

If you want to create identical host escalations on numerous hosts or hostgroups, but would like to exclude some hosts from the definition, this can be accomplished by preceding the host or hostgroup with a ! symbol.

```
define hostescalation{
    host_name             HOST1,HOST2,!HOST3,!HOST4,...,HOSTN
    hostgroup_name        HOSTGROUP1,HOSTGROUP2,!HOSTGROUP3,!HOSTGROUP4,...,HOSTGROUPN
    other escalation directives ...
}
```

7.21.7 Host Dependency Definitions

Multiple Hosts:

If you want to create *host dependencies* for multiple hosts, you can specify multiple hosts in the "host_name" and/or "dependent_host_name" directives. The definition below would be equivalent to creating six seperate host dependencies. In the example above, hosts HOST3, HOST4 and HOST5 would be dependent upon both HOST1 and HOST2. All the instances of the host dependencies would be identical except for the host names (i.e. have the same notification failure criteria, etc.).

```
define hostdependency{
    host_name               HOST1,HOST2
    dependent_host_name     HOST3,HOST4,HOST5
    other dependency directives ...
}
```

All Hosts In Multiple Hostgroups:

If you want to create host escalations for all hosts in in one or more hostgroups, you can do use the "hostgroup_name" and /or "dependent_hostgroup_name" directives. In the example below, all hosts in hostgroups HOSTGROUP3 and HOSTGROUP4 would be dependent on all hosts in hostgroups HOSTGROUP1 and HOSTGROUP2. All the instances of the host dependencies would be identical except for host names (i.e. have the same notification failure criteria, etc.).

```
define hostdependency{
    hostgroup_name                  HOSTGROUP1,HOSTGROUP2
    dependent_hostgroup_name        HOSTGROUP3,HOSTGROUP4
    other dependency directives ...
}
```

7.21.8 Hostgroups

All Hosts:

If you want to create a hostgroup that has all hosts that are defined in your configuration files as members, you can use a wildcard in the "members" directive. The definition below would create a hostgroup called HOSTGROUP1 that has all all hosts that are defined in your configuration files as members.

```
define hostgroup{
    hostgroup_name          HOSTGROUP1
    members                 *
    other hostgroup directives ...
}
```

7.22 Business rules

7.22.1 View your infrastructure from a business perspective

The main role of this feature is to allow users to have in one "indicator" the aggregation of other states. This indicator can provide a unique view for users focused on different roles.

Typical roles :

- Service delivery Management
- Business Management
- Engineering
- IT support

Let's take a simple example of a service delivery role for an ERP application. It mainly consists of the following IT components :

- 2 databases, in high availability, so with one database active, the service is considered up
- 2 web servers, in load sharing, so with one web server active, the service is considered up
- 2 load balancers, again in high availability

These IT components (Hosts in this example) will be the basis for the ERP service.

With business rules, you can have an "indicator" representing the "aggregated service" state for the ERP service! Shinken already checks all of the IT components one by one including processing for root cause analysis from a host and service perspective.

7.22.2 How to define Business Rules?

It's a simple service (or a host) with a "special" check_command named bp_rule. :)

This makes it compatible with all your current habits and UIs. As the service aggregation is considered as any other state from a host or service, you can get notifications, actions and escalations. This means you can have contacts that will receive only the relevant notifications based on their role.

> **Warning:** You do not have to define "bp_rule" command, it's purely internal. You should NOT define it in you checkcommands.cfg file, or the configuration will be invalid due to duplicate commands!

Here is a configuration for the ERP service example, attached to a dummy host named "servicedelivery".

```
define service{

   use         standard-service
   host_name   servicedelivery
   service_description  ERP
   check_command        bp_rule!(h1,database1 | h2,database2) & (h3,Http1 | h4,Http4) & (h5,IPVS1 | h
}
```

That's all!

> **Note:** A complete service delivery view should include an aggregated view of the end user availability perspective states, end user performance perspective states, IT component states, application error states, application performance states. This aggregated state can then be used as a metric for Service Management (basis for defining an SLA).

7.22.3 With "need at least X elements" clusters

In some cases, you know that in a cluster of N elements, you need at least X of them to run OK. This is easily defined, you just need to use the "X of:" operator.

Here is an example of the same ERP but with 3 http web servers, and you need at least 2 of them (to handle the load) :

```
define service{

   use         standard-service
   host_name   servicedelivery
   service_description  ERP
   check_command        bp_rule!(h1,database1 | h2,database2) & (2 of: h3,Http1 & h4,Http4 & h5,Http5
}
```

It's done :)

7.22.4 The NOT rule

You can define a not state rule. It can be useful for active/passive setups for example. You just need to add a ! before your element name.

For example :

```
define service{

   use         generic-service
   host_name   servicedelivery
   service_description  Cluster_state
   check_command        bp_rule!(h1,database1 & !h2,database2)
}
```

Aggregated state will be ok if database1 is ok and database2 is warning or critical (stopped).

7.22.5 Manage degraded status

In the Xof: way the only case where you got a "warning" (="degraded but not dead") it's when all your elements are in warning. But you should want to be in warning if 1 or your 3 http server is critical : the service is still running, but in a degraded state.

For this you can use the extended operator *X,Y,Zof:*

- X : number min of OK to get an overall OK state
- Y : number min of WARNING to get an overall WARNING state
- Z : number min of CRITICAL to get an overall CRITICAL state

State processing will be done the following order :

- is Ok possible?
- is critical possible?
- is warning possible?
- if none is possible, set OK.

Here are some example for business rules about 5 services A, B, C, D and E. Like 5,1,1of:A|B|C|D|E

Sample 1

A	B	C	D	E
Warn	Ok	Ok	Ok	Ok

Rules and overall states :

- 4of: –> Ok
- 5,1,1of: –> Warning
- 5,2,1of: –> Ok

Sample 2

A	B	C	D	E
Warn	Warn	Ok	Ok	Ok

Rules and overall states :

- 4of: –> Warning
- 3of: –> Ok
- 4,1,1of: –> Warning

Sample 3

A	B	C	D	E
Crit	Crit	Ok	Ok	Ok

Rules and overall states :

- 4of: –> Critical
- 3of: –> Ok
- 4,1,1of: –> Critical

Sample 4

A	B	C	D	E
Warn	Crit	Ok	Ok	Ok

Rules and overall states :

- 4of: –> Critical
- 4,1,1of: –> Critical

Sample 5

A	B	C	D	E
Warn	Warn	Crit	Ok	Ok

Rules and overall states :

- 2of: –> Ok
- 4,1,1of: –> Critical

Sample 6

A	B	C	D	E
Warn	Crit	Crit	Ok	Ok

Rules and overall states :

- 2of: –> Ok * 2,4,4of: –> Ok * 4,1,1of: –> Critical * 4,1,2of: –> Critical * 4,1,3of: –> Warning

Classic cases

Let's look at some classic setups, for MAX elements.

- ON/OFF setup : MAXof: <=> MAX,MAX,MAXof:
- Warning as soon as problem, and critical if all criticals : MAX,1,MAXof:
- Worse state : MAX,1,1

7.23 Migrating from Nagios to Shinken

7.23.1 How to to import existing Nagios states

It's possible with the *nagios_retention_file* module in fact.

The "migration" is done in two phases :

- First you launch shinken with both NagiosRetention and PickleRetention modules. It will load data from NagiosRetention and save them in a more "efficient" file. So add in *shinken-specififc.cfg* file both modules for your scheduler object:

```
modules                 NagiosRetention ,PickleRetention
```

- Then you remove the NagiosRetention (it's a read only module, don't fear for your nagios retention file) and restart with just PickleRetention. <code>modules PickleRetention

You're done.

//Source:// Topic on forum

Important: This method has met with limited success, further testing of the NagiosRetention module is required. An issues encountered should be raised in the Shinken issue tracker on github.

7.23.2 How to to import Nagios reporting data

There is no out of the box migration path for Historical reports.

7.24 Multi layer discovery

Shinken provides a discovery mecanism in several steps. There are on a side the **runners** (cf *Runners description*) which are script that output in formatted way properties list of scanned host and on another side discovery rules which use properties list to tag hosts when some of these properties are meaningful.

There are two kinds of rules, those which generate a host definition and those which launch another runners more specific to the scanned object. Better an image than a long speech :

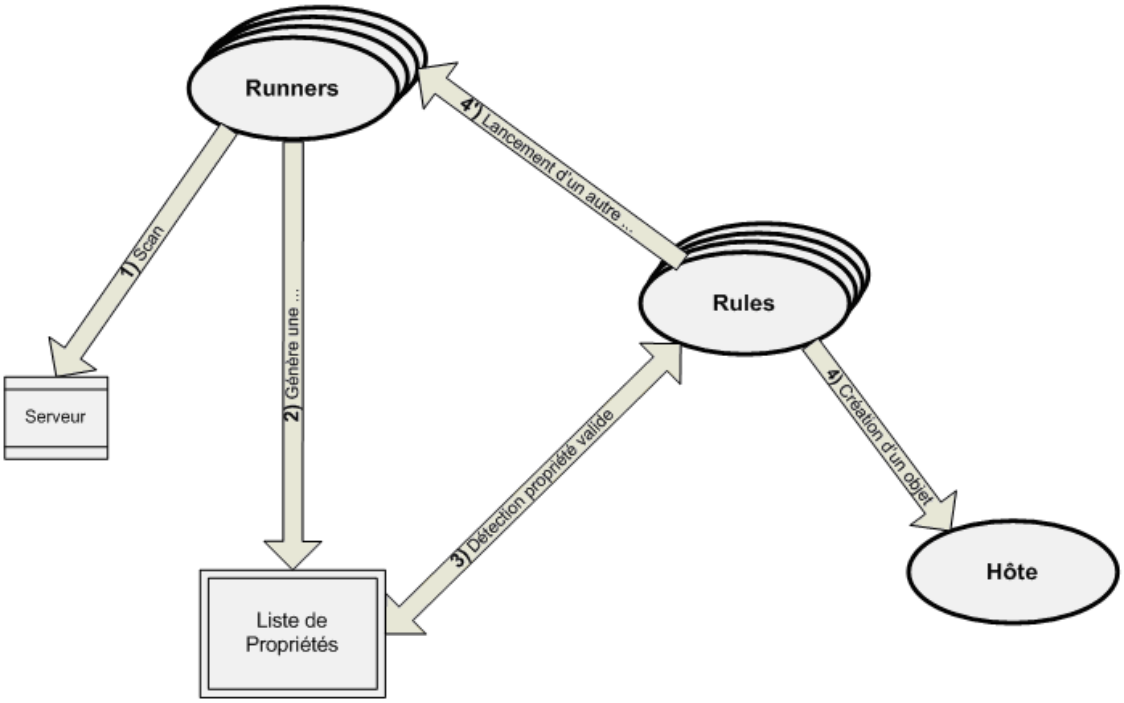

7.24.1 Runners available

Filesystems

Pre-requisites

To make this plugin works you must have snmp activated on targeted hosts. Take care to activate it and make HOST-RESSOURCES MIB OID available to it. Beginning OID of HOST-RESSOURCES MIB is : **.1.3.6.1.2.1.25**. The default discovery runner rule trigger this runner on unix host with port 161 open.

How it works:

FS discovery runner provides two modes : __macros__ and __tags__ modes. First one, __macros__ mode, will output a comma-separated list of filesystems under host macro '_fs', the other one will output tags with filesystems mountpoint.

Important: All filesystems will output with character **/** replaced by an underscore _.

Macros mode.

It is the easiest mode. It will add a line into host definition with host macro '_fs' with comma-separated list of filesystems. Then it is only needed to write a service definition using that macro with shinken directive "duplicate_foreach". Here is an example :

```
define service{
 service_description    Disks$KEY$
 use            generic-service
 register       0
 host_name      linux
 check_command  check_linux_disks!$KEY$

 duplicate_foreach    _fs
}
```

KEY will be replaced by '_fs' host macros value.

Tag mode

This mode will let you more flexibility to monitor filesystems. Each filesystems will be a tag named with filesystem mountpoint then you need discovery rules to tag scanned host with filesystem name. Example if you want to monitor "/var" filesystem on a host with following filesystems "/usr", "/var", "/opt", "/home", "/". You will need a discovery rules to match "/var", then a host template materializing the tag and a service applied to host template :

```
define discoveryrule {
      discoveryrule_name     fs_var
      creation_type          host
      fs                     var$
      +use                   fs_var
}
```

will match "/var" filesystem and tell to tag with "fs_var".

```
define host{
      name                   fs_var
      register               0
}
```

Host template used be scanned host.

```
define service{
      host_name              fs_var
      use                    10min_short
      service_description    Usage_var
      check_command          check_snmp_storage!"var$$"!50!25
      icon_set               disk
      register               0
}
```

and service applied to "fs_var" host template, itself applied to scanned host.

Now, if you want to apply same treatment to several filesystems, like "/var" and "/home" by example :

```
define discoveryrule {
      discoveryrule_name     fs_var_home
      creation_type          host
      fs                     var$|home$
```

```
      +use                    fs_var_home
}
```

```
define host{
      name                    fs_var_home
      register                0
}
```

```
define service{
      host_name               fs_var_home
      use                     10min_short
      service_description     Usage_var_and_home
      check_command           check_snmp_storage!"var$$|home$$"!50!25
      icon_set                disk
      register                0
}
```

Pay attention to double "$$", it is needed cause macros interpretation. When more than one "$" is used just double them else in this example we gotten Shinken trying to interprate macro '$|home$'.

Cluster

Pre-requisites

SNMP needed to make this runner works. You have to activate SNMP daemon on host discovered and make OID of clustering solution available to read. OID beginning for HACMP-MIB is : **.1.3.6.1.4.1.2.3.1.2.1.5.1** and for Safekit is : **.1.3.6.1.4.1.107.175.10**.

How it works

Runner does only detects HACMP/PowerHA and Safekit clustering solutions for the moment. It will scan OID and return cluster name or module name list, depends on Safekit or HACMP. For an host with two Safekit modules **test** and **prod**, runner will output :

```
# ./cluster_discovery_runnner.py -H sydlrtsm1 -O linux -C public
sydlrtsm1::safekit=Test,Prod
```

7.25 Multiple action urls

Since version <*insert the version number here*>, multiple action urls can be set for a service.

Syntax is :

```
define service {
 service_description name of the service
 [...]
 action_url URL1,ICON1,ALT1|URL2,ICON2,ALT2|URL3,ICON3,ALT3
}


* URLx are the url you want to use
* ICONx are the images you want to display the link as. It can be either a local file, relative to th
* ALTx are the alternative texts you want to display when the ICONx file is missing, or not set.
```

7.26 Aggregation rule

7.26.1 Goal

Got a way to define sort of agregation service for host services.

7.26.2 Sample 1

```
define host{
 _disks   /,/var,/backup
}

define service {
 register 0
 description  Disk $KEY$
 check_command   check_disk!$KEY$
}


define service {
 description All Disks
 check_command   bp_rule!., Disk $_HOSTDISKS$
}
```

ok this version sucks, we cannot parse this:

```
bp_rule!., Disk /,/var/backup</code>
```

7.26.3 version 2 (tag based agregation)

```
define host{
 name template
 register 0
}


define host{
 host_name host1
 use template
 _disks    /,/var,/backup
}


define service {
 register 0
 description  Disk $KEY$
 check_command   check_disk!$KEY$
 duplicate_foreach _disks
 business_rule_aggregate disks
}


define service {
 description All Disks
```

```
 host_name anyhost
 check_command   bp_rule!host1,a:disks
}


define service {
 description All Disks template based
 host_name template
 check_command   bp_rule!,a:disks
 register 0
}
```

7.27 Scaling Shinken for large deployments

7.27.1 Planning your deployment

A monitoring system needs to meet the expected requirements. The first thing you, as the system/network administrator need to do, is get management buy-in on deploying a supervisory and data acquisition system to meet corporate goals. The second is to define the scope of the monitoring system and its particularities.

- Number of services to supervise
- Service check frequency
- Method of supervising the services Passive versus Active
- Protocol used for data acquisition (ping, SSH, NSCA, TSCA, SNMP, NRPE, NSCAweb, collectd, scripts, etc)
- Retention duration for performance data
- Retention duration for status data
- For each status or performance data determine if it meets the scope and goals of the project.
- Can you live with interpolated data (RRD) or do you require exact representation of data (Graphite)
- Do you need to store performance data out of sequence (Graphite) or not (RRD)
- Do you need Active-Active HA for performance data (Graphite)
- Do you want to make use of Shinken's store and forward inter-process communications architecture to not lose performance data (Graphite) or not (RRD)

How scalable is Shinken

Shinken can scale out horizontally on multiple servers or vertically with more powerful hardware. Shinken deals automatically with distributed status retention. There is also no need to use external clustering or HA solutions.

Scalability can be described through a few key metrics

- Number of hosts + services supervised
- Number of active checks per second (type of active check having a major impact!)
- Number of check results per second (hosts and services combined)

And to a lesser extent, as performance data is not expected to overload a Graphite server instance (Which a single server can do up to 80K updates per second with millions of metrics) or even RRDTool+RRDcache with a hardware RAID 10 of 10K RPM disks.

- Number of performance data points (if using an external time-series database to store performance data)

Passive versus Active

Passive checks do not need to be scheduled by the monitoring server. Data acquisition and processing is distributed to the monitored hosts permitting lower acquisition intervals and more data points to be collected.

Active checks benefit from Shinken's powerful availability algorithms for fault isolation and false positive elimination.

A typical installation should make use of both types of checks.

Scaling the data acquisition

Thought needs to be used in determining what protocol to use and how many data points need to be collected will influence the acquisition method. There are many ways to slice an apple, but only a few scale beyond a few thousand services.

What is a big deployment? It depends on check frequency, number of services and check execution latency. 10K per minute NSCA based passive services is nothing for Shinken. 10K SSH checks per minute is unrealistic. 10K SNMP checks per minute can grind a server to a halt if not using an efficient polling method. Large deployments could easily ask for 20K, 50K, 80K services per minute.

Large numbers of active checks need to use poller modules

- nrpe_booster
- snmp_booster

Other integrated poller modules can be easily developed as required for ICMP(ping), SSH, TCP probes, etc.

Check_mk also uses a daemonized poller for its Windows and Unix agents which also makes it a good choice for scaling data acquisition from hosts. Note that WATO, the configuration front-end is not compatible with Shinken at this time. Check_mk is also limited to RRD backends.

Scaling the broker

The broker is a key component of the scalable architecture. Only a single broker can be active per scheduler. A broker can process broks (messages) from multiple schedulers. In most modern deployments, Livestatus is the broker module that provides status information to the web frontends. (Nagvis, Multisite, Thruk, etc.) or Shinken's own WebUI module. The broker needs memory and processing power.

Avoid using any broker modules that write logs or performance data to disk as an intermediate step prior to submission to the time series database. Use the Graphite broker module which will directly submit data to load-shared and/or redundant Graphite instances. Graphite is a time-series storage and retrieval database.

Make use of sqlite3 or mongodb to store Livestatus retention data. MongoDB integration with Livestatus is considered somewhat experimental, but can be very beneficial if performance and resiliency are desired. Especially when using a spare broker. MongoDB will ensure historical retention data is available to the spare broker, whereas with SQLite, it will not, and manual syncing is required.

Important: Code optimizations, a new JSON encoder/decoder, indexing and other means of decreasing access time to the in-memory data have been implemented in Shinken 1.2. These enhancements have improved markedly response time for small to extra large Livestatus instances.

Web Interface

MK Multisite and Nagvis are the only viable choices for very large installations. They can use multiple Nagios and Shinken monitoring servers as data providers and are based on the Livestatus API. Livestatus is a networked API for efficient remote access to Shinken run time data.

Dependancy model

Shinken has a great dependency resolution model. Automatic root cause isolation, at a host level, is one method that Shinken provides. This is based on explicitly defined parent/child relationships. This means that on a service or host failure, it will automatically reschedule an immediate check of the parent(s). Once the root failure(s) are found, any children will be marked as unknown status instead of soft down.

This model is very useful in reducing false positives. What needs to be understood is that it depends on defining a dependency **tree**. A dependency tree is restricted to single scheduler. Shinken provides a distributed architecture, that needs at least two trees for it to make sense.

Splitting trees by a logical grouping makes sense. This could be groups of services, geographic location, network hierarchy or other. Some elements may need to be duplicated at a host level (ex. ping check) like common critical elements (core routers, datacenter routers, AD, DNS, DHCP, NTP, etc.). A typical tree will involve clients, servers, network paths and dependent services. Make a plan, see if it works. If you need help designing your architecture, a professional services offering is in the works by the Shinken principals and their consulting partners.

Scaling the acquisition daemons

Typically pollers and Schedulers use up the most network, CPU and memory resources. Use the distributed architecture to scale horizontally on multiple commodity servers. Use at least a pair of Scheduler daemons on each server. Your dependency model should permit at least two trees, preferably 4.

7.27.2 Active acquisition methods

Scaling SNMP acquisition

Typically for networking devices, SNMP v2c is the most efficient method of data acquisition. Security considerations should be taken into account on the device accepting snmpv2c requests so that they are filtered to specific hosts and restricted to the required OIDs, this is device specific. Snmpv2c does not encrypt or protect the data or the passwords.

There is a myriad of SNMP monitoring scripts, most are utter garbage for scalable installations. This is simply due to the fact that every time they are launched a perl or python interpreter needs to be launched, modules need to be imported, the script executed, results get returned and then the script is cleared from memory. Rinse and repeat, very inefficient. Only two SNMP polling modules can meet high scalability requirements.

Shinken's integrated SNMP poller can scale to thousands of SNMP checks per second.

Check_mk also has a good SNMP acquisition model.

Scaling NRPE acquisition

Shinken provides an integrated NRPE check launcher. It is implemented in the poller as a module that allows to bypass the launch of the check_nrpe process. It reads the check command and opens the connection itself. It allows a big performance boost for launching check_nrpe calls.

The command definitions should be identical to the check_nrpe calls.

7.27.3 Passive acquisition methods

Scaling metric acquisition

Metrics or performance data (in Nagios speak) are embedded with check results. A check result can have zero or more performance metrics associated with it. Theses are transparently passed off to systems outside of Shinken using a Broker module. The Graphite broker module can easily send more than 2000 metrics per second. We have not tested the upper limit. Graphite itself can be configured to reach upper bounds of 80K metrics per second.

If a metric does not need its own service, it should be combined with a similar natured check being run on the server. Services are the expensive commodity, as they have all the intelligence like to them such as timeouts, retries, dependencies, etc. With Shinken 1.2 and fast servers, you should not exceed **60K services** for optimum performance.

Recommended protocols for scalable passive acquisition

- TSCA (Used under Z/OS and embedded in applications)
- Ws_Arbiter (Used by GLPI)
- NSCA (generic collection)
- Collectd (metrics only, states are calculated from metrics by the Shinen Scheduler using Shinken Python Triggers)

7.27.4 Log management methods

System and application logs should be gathered from servers and network devices. For this a centralized logging and analysis system is required.

Suggested centralized logging systems: OSSEC+Splunk for OSSEC, loglogic, MK Multisite log manager

Suggested windows agents:

- OSSEC agent
- Splunk universal forwarder

Suggested linux agent:

- OSSEC agent
- Splunk universal forwarder

Suggested Solaris agent:

- OSSEC agent
- Splunk universal forwarder

Splunk can aggregate the data, drop worthless data (unless mandated to log everything due to regulatory compliance), aggregate, analyze and alert back into Shinken. Log reporting and dashboards are a million times better in Splunk than anything else. If regulatory compliance causes too much data to be logged, look into using Kibana+logstash instead of Splunk, because Splunk costs a wicked lot per year.

7.27.5 SLA reporting methods

Feed Shinken event data back into Splunk, Thruk, Canopsis to get SLA reports. Use MK Multisites Livestatus based reporting.

7.27.6 Practical optimization tips

Chapter 59. Tuning Shinken For Maximum Performance

Internal Shinken metrics to monitor

7.28 Defining advanced service dependencies

First, the basics. You create service dependencies by adding service dependency definitions in your *object config file(s)*. In each definition you specify the *dependent* service, the service you are *depending on*, and the criteria (if any) that cause the execution and notification dependencies to fail (these are described later).

You can create several dependencies for a given service, but you must add a separate service dependency definition for each dependency you create.

7.28.1 Example Service Dependencies

The image below shows an example logical layout of service notification and execution dependencies. Different services are dependent on other services for notifications and check execution.

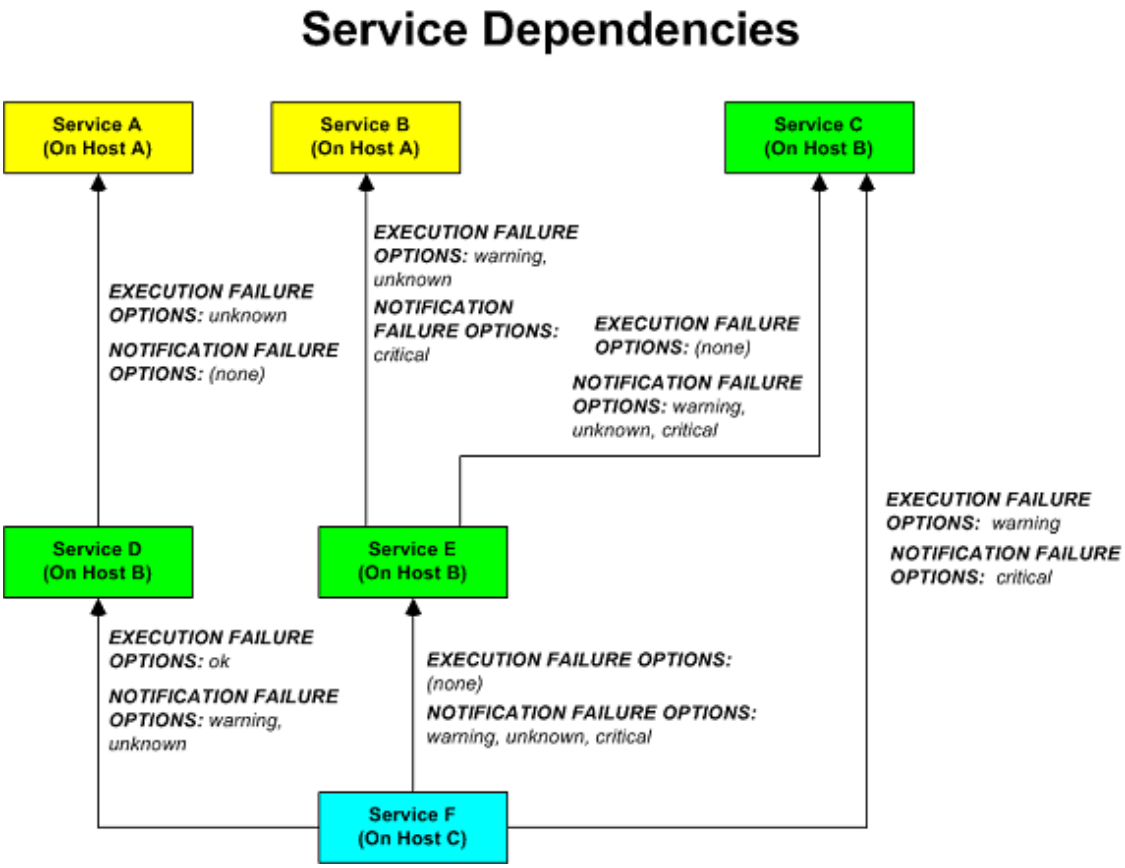

In this example, the dependency definitions for *Service F* on *Host C* would be defined as follows:

```
define servicedependency{
  host_name    Host B
  service_description    Service D
```

```
  dependent_host_name    Host C
  dependent_service_description    Service F
  execution_failure_criteria    o
  notification_failure_criteria    w,u
}

define servicedependency{
  host_name    Host B
  service_description    Service E
  dependent_host_name    Host C
  dependent_service_description    Service F
  execution_failure_criteria    n
  notification_failure_criteria    w,u,c
}

define servicedependency{
  host_name    Host B
  service_description    Service C
  dependent_host_name    Host C
  dependent_service_description    Service F
  execution_failure_criteria    w
  notification_failure_criteria    c
}
```

The other dependency definitions shown in the image above would be defined as follows:

```
define servicedependency{
  host_name    Host A
  service_description    Service A
  dependent_host_name    Host B
  dependent_service_description    Service D
  execution_failure_criteria    u
  notification_failure_criteria    n
}

define servicedependency{
  host_name    Host A
  service_description    Service B
  dependent_host_name    Host B
  dependent_service_description    Service E
  execution_failure_criteria    w,u
  notification_failure_criteria    c
}

define servicedependency{
  host_name    Host B
  service_description    Service C
  dependent_host_name    Host B
  dependent_service_description    Service E
  execution_failure_criteria    n
  notification_failure_criteria    w,u,c
}
```

7.28.2 How Service Dependencies Are Tested

Before Shinken executes a service check or sends notifications out for a service, it will check to see if the service has any dependencies. If it doesn't have any dependencies, the check is executed or the notification is sent out as it

normally would be. If the service *does* have one or more dependencies, Shinken will check each dependency entry as follows:

- Shinken gets the current status:ref:* *<advancedtopics-dependencies#advancedtopics-dependencies-hard_dependencies>* of the service that is being *depended upon*.
- Shinken compares the current status of the service that is being *depended upon* against either the execution or notification failure options in the dependency definition (whichever one is relevant at the time).
- If the current status of the service that is being *depended upon* matches one of the failure options, the dependency is said to have failed and Shinken will break out of the dependency check loop.
- If the current state of the service that is being *depended upon* does not match any of the failure options for the dependency entry, the dependency is said to have passed and Shinken will go on and check the next dependency entry.

This cycle continues until either all dependencies for the service have been checked or until one dependency check fails.

- One important thing to note is that by default, Shinken will use the most current *hard state* of the service(s) that is/are being depended upon when it does the dependency checks. If you want Shinken to use the most current state of the services (regardless of whether its a soft or hard state), enable the soft_state_dependencies option.

7.28.3 Execution Dependencies

Execution dependencies are used to restrict when *active checks* of a service can be performed. *Passive checks* are not restricted by execution dependencies.

If all of the execution dependency tests for the service passed, Shinken will execute the check of the service as it normally would. If even just one of the execution dependencies for a service fails, Shinken will temporarily prevent the execution of checks for that (dependent) service. At some point in the future the execution dependency tests for the service may all pass. If this happens, Shinken will start checking the service again as it normally would. More information on the check scheduling logic can be found *here*.

In the example above, **Service E** would have failed execution dependencies if **Service B** is in a WARNING or UNKNOWN state. If this was the case, the service check would not be performed and the check would be scheduled for (potential) execution at a later time.

> **Warning:** Execution dependencies will limit the load due to useless checks, but can limit some correlation logics, and so should be used only if you trully need them.

7.28.4 Notification Dependencies

If all of the notification dependency tests for the service *passed*, Shinken will send notifications out for the service as it normally would. If even just one of the notification dependencies for a service fails, Shinken will temporarily repress notifications for that (dependent) service. At some point in the future the notification dependency tests for the service may all pass. If this happens, Shinken will start sending out notifications again as it normally would for the service. More information on the notification logic can be found *here*.

In the example above, **Service F** would have failed notification dependencies if **Service C** is in a CRITICAL state, //and/or* **Service D** is in a WARNING or UNKNOWN state, *and/or// if **Service E** is in a WARNING, UNKNOWN, or CRITICAL state. If this were the case, notifications for the service would not be sent out.

7.28.5 Dependency Inheritance

As mentioned before, service dependencies are not inherited by default. In the example above you can see that Service F is dependent on Service E. However, it does not automatically inherit Service E's dependencies on Service B and Service C. In order to make Service F dependent on Service C we had to add another service dependency definition. There is no dependency definition for Service B, so Service F is not dependent on Service B.

If you do wish to make service dependencies inheritable, you must use the inherits_parent directive in the service dependency definition. When this directive is enabled, it indicates that the dependency inherits dependencies of the service that is being depended upon (also referred to as the master service). In other words, if the master service is dependent upon other services and any one of those dependencies fail, this dependency will also fail.

In the example above, imagine that you want to add a new dependency for service F to make it dependent on service A. You could create a new dependency definition that specified service F as the dependent service and service A as being the master service (i.e. the service that is being dependend on). You could alternatively modify the dependency definition for services D and F to look like this:

```
define servicedependency{
  host_name    Host B
  service_description    Service D
  dependent_host_name    Host C
  dependent_service_description    Service F
  execution_failure_criteria    o
  notification_failure_criteria    n
  inherits_parent    1
}
```

Since the inherits_parent directive is enabled, the dependency between services A and D will be tested when the dependency between services F and D are being tested.

Dependencies can have multiple levels of inheritance. If the dependency definition between A and D had its inherits_parent directive enable and service A was dependent on some other service (let's call it service G), the service F would be dependent on services D, A, and G (each with potentially different criteria).

7.28.6 Host Dependencies

As you'd probably expect, host dependencies work in a similar fashion to service dependencies. The difference is that they're for hosts, not services.

Do not confuse host dependencies with parent/child host relationships. You should be using parent/child host relationships (defined with the parents directive in host definitions) for most cases, rather than host dependencies. A description of how parent/child host relationships work can be found in the documentation on *network reachability*.

Here are the basics about host dependencies:

- A host can be dependent on one or more other host
- Host dependencies are not inherited (unless specifically configured to)
- Host dependencies can be used to cause host check execution and host notifications to be suppressed under different circumstances (UP, DOWN, and/or UNREACHABLE states)
- Host dependencies might only be valid during specific *timeperiods*

7.28.7 Example Host Dependencies

The image below shows an example of the logical layout of host notification dependencies. Different hosts are dependent on other hosts for notifications.

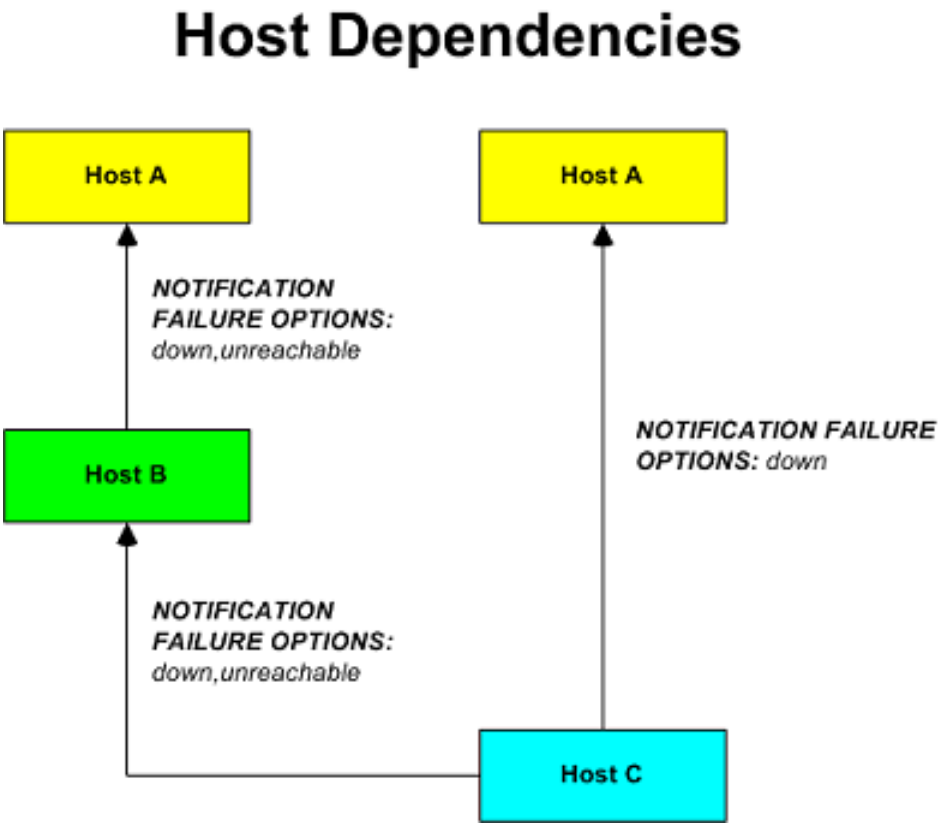

In the example above, the dependency definitions for Host C would be defined as follows:

```
define hostdependency{
  host_name    Host A
  dependent_host_name    Host C
  notification_failure_criteria    d
}

define hostdependency{
  host_name    Host B
  dependent_host_name    Host C
  notification_failure_criteria    d,u
}
```

As with service dependencies, host dependencies are not inherited. In the example image you can see that Host C does not inherit the host dependencies of Host B. In order for Host C to be dependent on Host A, a new host dependency definition must be defined.

Host notification dependencies work in a similar manner to service notification dependencies. If *all* of the notification dependency tests for the host *pass*, Shinken will send notifications out for the host as it normally would. If even just one of the notification dependencies for a host fails, Shinken will temporarily repress notifications for that (dependent) host. At some point in the future the notification dependency tests for the host may all pass. If this happens, Shinken will start sending out notifications again as it normally would for the host. More information on the notification logic can be found *here*.

7.29 Shinken's distributed architecture

7.29.1 Shinken's distributed architecture for load balancing

The load balancing feature is very easy to obtain with Shinken. If I say that the project's name comes from it you should believe me :)

If you use the distributed architecture for load balancing, know that load is typically present in 2 processes:

- pollers: they launch checks, they use a lot of CPU resources
- schedulers: they schedule, potentially lots of checks

For both, a limit of 150000 checks/5min is a reasonable goal on an average server(4 cores@3Ghz). Scaling can be achieved horizontally by simply adding more servers and declaring them as pollers or schedulers.

Tip: The scheduler is NOT a multi-threaded process, so even if you add cores to your server, it won't change it's performances.

There are mainly two cases where load is a problem:

- using plugins that require lots of processing (check_esx3.pl is a good example)
- scheduling a very large number of checks (> 150000 checks in 5 minutes).

In the first case, you need to add more pollers. In the second, you need to add more schedulers. In this last case, you should also consider adding more pollers (more checks = more pollers) but that can be determined by the load observed on your poller(s).

From now, we will focus on the first case, typically installations have less than 150K checks in 5 minutes, and will only need to scale pollers, not schedulers.

7.29.2 Setup a load balancing architecture with some pollers

Install the poller on the new server

But I already hear you asking "How to add new satellites?". That's very simple: **you start by installing the application on a new server like you did in the 10 min starting tutorial but you can pass the discovery, the webUI and skip the /etc/init.d/shinken script (or Windows services)**.

Let say that this new server is called server2 and has the IP 192.168.0.2 and the "master" is called server1 with 192.168.0.1 as its IP.

Tip: You need to have all plugins you use in server1 also installed on server2, this should already done if you followed the 10 min tutorial.

On server2, you just need to start the poller service, not the whole Shinken stack.

```
 On ubuntu/debian:
update-rc.d shinken-poller default
 On RedHat/Centos:
chkconfig --add shinken-poller
chkconfig shinken-poller on
```

Then start it:

```
sudo /etc/init.d/shinken-poller start
```

Warning: DO NOT START the arbiter on the server2 for load balancing purpose. It can be done for high availability. Unless you know what you are doing, don't start the arbiter process!^_^

Declare the new poller on the main configuration file

Ok, now you have a brand new poller declared on your new server, server2. **But server1 needs to know that it must give work to it. This is done by declaring the new poller in the shinken-specific.cfg file.**

Edit your /etc/shinken-specific.cfg file (or c:shinkenetcshinken-specific.cfg under Windows) and define your new poller under the existing poller-1 definition (on server1):

```
#Pollers launch checks
define poller{
     poller_name      poller-2
     address          server2
     port             7771
}
```

Be sure to have also those lines:

```
define scheduler{
     scheduler_name scheduler-1 ; just the name
     address  192.168.0.1           ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon
```

The address has to be 192.168.0.1 or server1 but not localhost!

Important: Check that the line named host in the scheduler.ini is 0.0.0.0 in order to listen on all interfaces.

When it's done, restart your arbiter:

```
Under Linux:
sudo /etc/init.d/shinken-arbiter restart
Under Windows:
net stop shinken-arbiter
net start shinken-arbiter
```

It's done! You can look at the global shinken.log file (should be under /var/lib/shinken/shinken.log or c:shinkenvarshinken.log) that the new poller is started and can reach scheduler-1. So look for lines like:

```
[All] poller satellite order: poller-2 (spare:False), poller-1 (spare:False),
[All] Trying to send configuration to poller poller-2
[All] Dispatch OK of for configuration 0 to poller poller-2
```

You can also look at the poller logs on server2. You may have lines like that:

```
Waiting for initial configuration
[poller-2] Init de connection with scheduler-1 at PYROLOC://192.168.0.1:7768/Checks
[poller-2] Connexion OK with scheduler scheduler-1
We have our schedulers: {0: {'wait_homerun': {}, 'name': u'scheduler-1', 'uri': u'PYROLOC://192.168.0
I correctly loaded the modules: []
[poller-2] Allocating new fork Worker: 0
```

7.30 Shinken's distributed architecture with realms

7.30.1 Multi customers and/or sites: REALMS

Shinken's architecture like we saw allows us to have a unique administration and data location. All pollers the hosts are cut and sent to schedulers, and the pollers take jobs from all schedulers. Every one is happy.

Every one? In fact no. If an administrator got a continental distributed architecture he can have serious problems. If the architecture is common to multiple customers network, a customer A scheduler can have a customer B poller that asks him jobs. It's not a good solution. Even with distributed network, distant pollers should not ask jobs to schedulers in the other continent, it's not network efficient.

That is where the site/customers management is useful. In Shinken, it's managed by the **realms**.

A realm is a group of resources that will manage hosts or hostgroups. Such a link will be unique: a host cannot be in multiple realms. If you put an hostgroup in a realm, all hosts in this group will be in the realm (unless a host already has the realm set, the host value will be taken).

A realm is:

- at least a scheduler
- at least a poller
- can have a reactionner
- can have a broker

In a realm, all realm pollers will take all realm schedulers jobs.

Important: Very important: there is only ONE arbiter (and a spare of course) for ALL realms. The arbiter manages all realms and all that is inside.

7.30.2 Sub-realms

A realm can have sub-realms. It doesn't change anything for schedulers, but it can be useful for other satellites and spares. Reactionners and brokers are linked to a realm, but they can take jobs from all sub-realms too. This way you can have less reactionners and brokers (like we soon will see).

The fact that reactionners/brokers (and in fact pollers too) can take jobs from sub-schedulers is decided by the presence of the manage_sub_realms parameter. For pollers the default value is 0, but it's 1 for reactionners/brokers.

7.30.3 An example

To make it simple: you put hosts and/or hostgroups in a realm. This last one is to be considered as a resources pool. You don't need to touch the host/hostgroup definition if you need more/less performances in the realm or if you want to add a new satellites (a new reactionner for example).

Realms are a way to manage resources. They are the smaller clouds in your global cloud infrastructure :)

If you do not need this feature, that's not a problem, it's optional. There will be a default realm created and every one will be put into.

It's the same for hosts that don't have a realm configured: they will be put in the realm that has the "default" parameter.

7.30.4 Picture example

Diagrams are good :)

Let's take two examples of distributed architectures around the world. In the first case, the administrator don't want to share resources between realms. They are distinct. In the second, the reactionners and brokers are shared with all realms (so all notifications are send from a unique place, and so is all data).

Here is the isolated one:

And a more common way of sharing reactionner/broker:

Like you can see, all elements are in a unique realm. That's the sub-realm functionality used for reactionner/broker.

7.30.5 Configuration of the realms

Here is the configuration for the shared architecture:

```
define realm {
 realm_name       All
 realm_members    Europe,US,Asia
 default          1    ;Is the default realm. Should be unique!
}


define realm{
 realm_name       Europe
 realm_members    Paris   ;This realm is IN Europe
}
```

An now the satellites:

```
define scheduler{
 scheduler_name       scheduler_Paris
 realm                Paris             ;It will only manage Paris hosts
}


define reactionner{
 reactionner_name     reactionner-master
 realm                All                ;Will reach ALL schedulers
}
```

And in host/hostgroup definition:

```
define host{
 host_name         server-paris
 realm             Paris         ;Will be put in the Paris realm
 [...]
}


define hostgroups{
 hostgroup_name                linux-servers
 alias                         Linux Servers
 members                       srv1,srv2
```

```
 realm                        Europe       ;Will be put in the Europe realm
}
```

7.30.6 Multi levels brokers

In the previous samples, if you put numerous brokers into the realm, each scheduler will have only one broker at the same time. It was also impossible to have a common Broker in All, and one brokers in each sub-realms.

You can activate multi-brokers features with a realm parameter, the broker_complete_links option (0 by default).

You will have to enable this option in ALL your realms! For example:

```
define realm{
 realm_name       Europe
 broker_complete_links  1
}
```

This will enable the fact that each scehduler will be linked with each brokers. This will make possible to have dedicated brokers in a same realm (one for WebUI, another for Graphite for example). It will also make possible to have a common Broker in "All", and one broker in each of its sub-realms (Europe, US and Asia). Of course the sub-brokers will only see the data from their realms, and the sub-realms (like Paris for Europe for example).

7.31 Macro modulations

7.31.1 How macros modulations works

It's a good idea to have macros for critical/warning levels on the host or its templates. But sometime even with this, it can be hard to manage such cases wher you want to have high levels during the night, and a lower one during the day.

macro_modulations is made for this.

7.31.2 How to define a macro_modulation

```
define  macromodulation{
      macromodulation_name            HighDuringNight
      modulation_period               night
      _CRITICAL                       20
      _WARNING                        10
}


define host{
  check_command                  check_ping
  check_period                   24x7
  host_name                      localhost
  use                            generic-host
  macromodulations               HighDuringNight
  _CRITICAL                      5
  _WARNING                       2
}
```

With this, the services will get 5 and 2 for the threshold macros during the day, and will automatically get 20 and 10 during the night timeperiod. You can have as many modulations as you want. **The first modulation enabled will take the lead.**

7.32 Shinken and Android

Shinken can run on an android device like a phone. It can be very useful for one particular daemon: the reactionner that send alerts. With this, you can setup a "sms by phone" architecture, with high availability. We will see that you can also receive ACKs by SMS :)

All you need is one (or two if you want high availability) android phone with an internet connection and Wifi. Any version should work.

Tip: This is of course for fun. Unless you have a secure connection to your monitoring infrastructure. You should never open up your firewall to have in/out communications from a mobile phone directly to your monitoring systems. A serious infrastructure should use an SMS gateway in a DMZ that receives notifications from a your monitoring system. Either sourced as mails, or other message types.

7.32.1 Sending SMS

Install Python on your phone

- enable the "Unknown sources" option in your device's "Application" settings to allow application installation from another source that the android marker.
- Go to http://code.google.com/p/android-scripting/ and "flash" the barcode with an application like "barcode scanner", or just download http://android-scripting.googlecode.com/files/sl4a_r4.apk. Install this application.
- Launch the sl4a application you just installed.
- click in the menu button, click "view" and then select "interpreter"
- click the menu button again, then add and select "Python 2.6". Then click to install.

Install the Pyro lib on your phone

Go to http://pypi.python.org/pypi/Pyro4/ and download the same Pyro that you are using in Shinken.

- Connect your phone to a computer, and open the sdcard disk.
- Untar the Pyro4 tar ball, and copy the Pyro4 library directory (the one IN the Pyro4-10 directory, NOT the 4.10 directory itself) and copy/paste it in SDCARD/com.googlecode.pythonforandroidextraspython directory. Be sure the file SDCARDcom.googlecode.pythonforandroidextraspythonPyro__init__.py exists, or you put the wrong directory here.
- Don't close your sdcard explorer

Install Shinken on your phone

- Like for Pyro, copy your shinken library directory in SDCARDcom.googlecode.pythonforandroidextraspython. If you do not have the SDCARDcom.googlecode.pythonforandroidextraspythonshinken__ini__.py file, you put the bad directory.
- Copy the bin/shinken-reactionner file in SDCARDsl4ascripts direcotry and rename it shinken-reactionner.py (so add the .py extension)

Time to launch the Shinken app on the phone

- Unmount the phone from your computer and be sure to re-mount the sdcard on your phone (look at the notifications).
- Launch the sl4a app
- launch the shinken-reactionner.py app in the script list.
- It should launch without errors

Declare this daemon in the central configuration

The phone(s) will be a new reactionner daemon. You should want to only launch SMS with it, not mail commands or nother notifications. So you will have to define this reactionner to manage only the SMS commands. There is an example of such SMS-reactionner in the sample etc/shinken-specific.cfg file and the module AndroidSMS need by this reactionner to send SMS with android.

```
define reactionner{
     reactionner_name             reactionner-Android
     address                      WIFIIPOFYOURPHONE
     port                         7769
     spare                        0

     # Modules
     modules         AndroidSMS
     reactionner_tags        android_sms

     }


# Reactionner can be launched under an android device
# and can be used to send SMS with this module
define module{
     module_name      AndroidSMS
     module_type      android_sms
}
```

The important lines are:

- address: put the Wifi address of your phone
- modules: load the Android module to be able to manage sms sent.
- reactionner_tags: only android_sms commands will be send to this reactionner.

In the commands.cfg, there are example of sms sending commands

```
# For Android SMS things
# You need both reactionner_tag and module_type in most cases!
define command{
     command_name                    notify-host-by-android-sms
     command_line                    android_sms  $CONTACTPAGER$ Host: $HOSTNAME$\nAddress: $HOSTADDR
     reactionner_tag                 android_sms
     module_type                     android_sms
}


define command{
     command_name                    notify-service-by-android-sms
```

```
     command_line                    android_sms  $CONTACTPAGER$ Service: $SERVICEDESC$\nHost: $HOSTN
     reactionner_tag                 android_sms
     module_type                     android_sms
}
```

The important part are the reactionner_tag and module_type lines. With this parameter, you are sure the command will be mar

- only the reactionner(s) with the tag android_sms, and in this reactionner, it will be managed by the module android_sms.

Add SMS notification ways

In order to use SMS, it is a good thing to add notification way dedicated to send SMS, separated from email notifications. Edit templates and add theses lines to declare a new notification way using SMS (*more about notification ways*) :

```
define notificationway{
     notificationway_name            android-sms
     service_notification_period     24x7
     host_notification_period        24x7
     service_notification_options    c,w,r
     host_notification_options       d,u,r,f,s
     service_notification_commands   notify-service-by-android-sms
     host_notification_commands      notify-host-by-android-sms
}
```

Add SMS to your contacts

You only need to add theses commands to your contacts (or contact templates, or notification ways) to send them SMS:

```
define contact{
      name                            generic-contact         ; The name of this contact template
      [...]
      notificationways                email,android-sms       ; Use email and sms to notify the conta
```

That's all.

7.32.2 Receive SMS: acknowledge with a SMS

Pre-requite

You need to have a working android-reactionner with the sms module. The sms reception will be automatically enabled.

How to send ACK from SMS?

All you need is to send a SMS to the phone with the format:

For a service:

```
ACK  host_name/service_description
```

For an host:

```
ACK  host_name
```

And it will automatically raise an acknowledgment for this object :)

7.33 Send sms by gateway

Shinken can be used to send sms to you and other people when you got an alert.

I will tell you how to do it with ovh gateway. If you need for another one you need to modify a little bit the information.

7.33.1 1. you need to go to your contact.cfg who is for linux in /usr/local/shinken/etc/contacts.cfg

For each user you need to add her phone number in the pager line. (For ovh you need to do it with 0032 for example and not +32 , all phone number must be with the international prefix).

In the same file you need also to add these lines in each contact you want that I receive ams.

```
host_notifications_enabled      1                    // This will activate the notifications for the
service_notifications_enabled   1                    // This will activate the notifications for the
notificationways SMS                                 // This is the name of your notifications ways
```

Then you need to add this at the end of the contacts.cfg

```
define notificationway{
     notificationway_name            SMS           // Here you need to put the name of the notificati
     service_notification_period     24x7          // Here I will receive ams all the time, If you wa
     host_notification_period        24x7          // Same as above
     service_notification_options    w,c,r         // It tell that I want receive a sms for the hosts
    host_notification_options         d,r          // It tell that I want receive a sms for the servi
     service_notification_commands   notify-service-by-ovhsms // The name of the notifications
     host_notification_commands      notify-host-by-ovhsms
}
```

7.33.2 2. you need to go to your commands.cfg who is in /usr/local/shinken/etc/commands.cfg

And add these line at the end.

```
# Notify Service by SMS-OVH
define command {
  command_name        notify-service-by-ovhsms     // Should be the same as in the contacts.cfg
  command_line        $PLUGINSDIR$/ovhsms.sh  $CONTACTPAGER$ $NOTIFICATIONTYPE$ $SERVICEDESC$ $HOSTNA
}


# Notify host by SMS-OVH
define command {
  command_name        notify-host-by-ovhsms      * * Should be the same as in the contacts.cfg
  command_line        $PLUGINSDIR$/ovhsms.sh $CONTACTPAGER$ $NOTIFICATIONTYPE$ $SERVICEDESC$ $HOSTNAM
}
```

7.33.3 3. Add the script

First you need to be the shinken user so do a : su shinken do a : cd /usr/local/shinken/libexec/ and then create and edit your new script with the name you set above : nano -w ovhsms.sh

```
#!/bin/bash


date > ~/datesms

NOTIFICATIONTYPE=$2
HOSTALIAS=$3
SERVICEDESC=$4
SERVICESTATE=$5
textesms="**"$NOTIFICATIONTYPE" alerte - "$HOSTALIAS"/"$SERVICEDESC" is "$SERVICESTATE" **" // This i
wget -o ~/logenvoisms -O ~/reponse "https://www.ovh.com/cgi-bin/sms/http2sms.cgi?smsAccount=sms-XXXX

exit 0
```

7.33.4 4. Test It

Save your file and do : "exit" To exit the shinken user. Then set down one of your host or service to test if you receive it.

7.34 Triggers

Warning: This is currently in Beta. DO NOT use in production.

Note: Not up to date. See the bottom part for something recent

A trigger object is something that can be called after a "change" on an object. It's a bit like Zabbix trigger, and should be used only if you need it. In most cases, direct check is easier to setup :)

It's defined like:

Here is an example that will raise a critical check if the CPU is too loaded:

7.34.1 Simple rule

```
define trigger{
 trigger_name    One_Cpu_too_high
 matching_rule   perf(self, 'cpu') >= 95
 hit_action      critical(self, 'Cpu is too loaded')
}
```

7.34.2 Rule with an OR

Another one that will look if at least one CPU is too loaded (> 90% load) or the overall CPU is too loaded too (total > 60%):

```
define trigger{
 trigger_name    One_or_more_cpu_too_high
 matching_rule   max([perf(self, 'cpu*')]) > 90 | avg([perf(self, 'cpu*')]) > 60
 hit_action      critical(self, 'Cpu is too loaded')
}
```

7.34.3 Advanced correlation: active/passive cluster check

It can be used for advanced correlation too:

If you want to do an active/passive check without a bp_rule here an example. This service will be the "cluster" service that show the overall state. It will have 2 custom macros: "master", the master server and "slave" the slave one.

```
define trigger{
 trigger_name    Bad_active_passive
 matching_rule   (service(self.customs['master']).state == 'CRITICAL' & service(self.customs['slave']
 hit_action      critical(self, 'Cluster got a problem')
}
```

And if you want you can define a degraded one you can define another trigger for this same "cluster" service:

```
define trigger{
 trigger_name    Degraded_service
 matching_rule   service(self.customs['master']).state == 'CRITICAL' & service(self.customs['slave']
 hit_action      warning(self, 'Cluster runs on slave!')
}
```

7.34.4 Statefull rules

Here an example with statefull rules.

I will read a regexp like PORTSCAN FROM (S+) TO S+:(d+) on a service, and create an "event" that got a 60min lifetime. It will be add on services on all hosts for example.

```
define trigger{
 trigger_name    Log_post_scan
 matching_rule   regexp(self.output, 'PORTSCAN FROM (?P<source>\S+) TO (?P<dest>\S+):(?P<port>\d+)')
 hit_action      create_event('HORIZONTAL SCAN FROM SOURCE IP %s' % source, 60)
}
```

And a aggregated one will raise the alert if need:

```
define trigger{
 trigger_name    Raise_too_much_scans
 matching_rule   sources=get_events_count_group_by('HORIZONTAL SCAN FROM SOURCE IP (?P<source>\S+)')
 hit_action      [critical(self, 'The IP %s scan too much ips' % source) for (source, nb) in sources
}
```

7.34.5 Compute KPI

You maybe want to compute a "KPI" (key point indicator) from various sources. You can also do it with triggers.

Let take an example, You got a cluster of N webservers. Each is returning in a check the number of active connections, but you want the overall. You just need to define a new service that will take it's data from the N others.

```
define trigger{
 trigger_name    Count_active_connections
 matching_rule   True;total_connections=sum(perfs('web-srv*/Http', 'active_connections'))
 hit_action      set_perfdata(self, 'total_connections=%d' % total_connections)
}
```

7.34.6 Define and use triggers

Note: More or less up to date

Use the trigger_name directive to link a trigger to a service or host. Example :

```
define service{
      use                             local-service         ; Name of service template to use
      host_name                       localhost
      service_description             Current Load trigger
      check_command                   check_local_load!5.0,4.0,3.0!10.0,6.0,4.0
      trigger_name                    simple_cpu
      }
```

Then define your trigger in etc/trigger.d/yourtrigger.trig. here the file is simple_cpu.trig

```
try:

  load = perf(self, 'load1')
  print "Founded load", load
  if load >= 10:
      critical(self, 'CRITICAL | load=%d' % load)
  elif load >= 5:
      warning(self, 'WARNING | load=%d' % load)
  else:
      ok(self, 'OK | load=%d' % load)
except:

  unknown(self, 'UNKNOWN | load=%d' % load)
```

Finally, add the triggers_dir=trigger.d statement to your shinken.cfg

7.35 Unused nagios parameters

The parameters below are managed in Nagios but not in Shinken because they are useless in the architecture. If you really need one of them, please use Nagios instead or send us a patch :)

Note: The title is quite ambiguous : a not implemented parameter is different from an unused parameter.

The difference has been done in this page, why about creating a not_implemented_nagios_parameters?

7.35.1 External Command Check Interval (Unused)

Format:	command_check_interval=<xxx>[s]
Example:	command_check_interval=1

If you specify a number with an "s" appended to it (i.e. 30s), this is the number of seconds to wait between external command checks. If you leave off the "s", this is the number of "time units" to wait between external command checks. Unless you've changed the *Timing Interval Length* value (as defined below) from the default value of 60, this number will mean minutes.

By setting this value to **-1**, Nagios will check for external commands as often as possible. Each time Nagios checks for external commands it will read and process all commands present in the *External Command File* before continuing on with its other duties. More information on external commands can be found *here*.

7.35.2 External Command Buffer Slots (Not implemented)

Format:	external_command_buffer_slots=<#>
Example:	external_command_buffer_slots=512

This is an advanced feature.

This option determines how many buffer slots Nagios will reserve for caching external commands that have been read from the external command file by a worker thread, but have not yet been processed by the main thread of the Nagios deamon. Each slot can hold one external command, so this option essentially determines how many commands can be buffered. For installations where you process a large number of passive checks (e.g. *distributed setups*), you may need to increase this number. You should consider using MRTG to graph Nagios' usage of external command buffers.

7.35.3 Use Retained Program State Option (Not implemented)

Format:	use_retained_program_state=<0/1>
Example:	use_retained_program_state=1

This setting determines whether or not Nagios will set various program-wide state variables based on the values saved in the retention file. Some of these program-wide state variables that are normally saved across program restarts if state retention is enabled include the *Notifications Option*, *Flap Detection Option*, *Event Handler Option*, *Service Check Execution Option*, and *Passive Service Check Acceptance Option* !!!!!!!!!! options. If you do not have *State Retention Option* enabled, this option has no effect.

- 0 = Don't use retained program state
- 1 = Use retained program state (default)

7.35.4 Use Retained Scheduling Info Option (Not implemented)

Format:	use_retained_scheduling_info=<0/1>
Example:	use_retained_scheduling_info=1

This setting determines whether or not Nagios will retain scheduling info (next check times) for hosts and services when it restarts. If you are adding a large number (or percentage) of hosts and services, I would recommend disabling this option when you first restart Nagios, as it can adversely skew the spread of initial checks. Otherwise you will probably want to leave it enabled.

- 0 = Don't use retained scheduling info
- 1 = Use retained scheduling info (default)

7.35.5 Retained Host and Service Attribute Masks (Not implemented)

Format:	retained_host_attribute_mask=<number> retained_service_attribute_mask=<number>
Example:	retained_host_attribute_mask=0 retained_service_attribute_mask=0

This is an advanced feature. You'll need to read the Nagios source code to use this option effectively.

These options determine which host or service attributes are NOT retained across program restarts. The values for these options are a bitwise AND of values specified by the "**MODATTR_**" definitions in the "include/common.h" source code file. By default, all host and service attributes are retained.

7.35.6 Retained Process Attribute Masks (Not implemented)

Format:	retained_process_host_attribute_mask=<number> retained_process_service_attribute_mask=<number>
Example:	retained_process_host_attribute_mask=0 retained_process_service_attribute_mask=0

This is an advanced feature. You'll need to read the Nagios source code to use this option effectively.

These options determine which process attributes are NOT retained across program restarts. There are two masks because there are often separate host and service process attributes that can be changed. For example, host checks can be disabled at the program level, while service checks are still enabled. The values for these options are a bitwise AND of values specified by the "**MODATTR_**" definitions in the "include/common.h" source code file. By default, all process attributes are retained.

7.35.7 Retained Contact Attribute Masks (Not implemented)

Format:	retained_contact_host_attribute_mask=<number> retained_contact_service_attribute_mask=<number>
Example:	retained_contact_host_attribute_mask=0i retained_contact_service_attribute_mask=0

This is an advanced feature. You'll need to read the Nagios source code to use this option effectively.

These options determine which contact attributes are NOT retained across program restarts. There are two masks because there are often separate host and service contact attributes that can be changed. The values for these options are a bitwise AND of values specified by the "**MODATTR_**" definitions in the "include/common.h" source code file. By default, all process attributes are retained.

7.35.8 Service Inter-Check Delay Method (Unused)

Format:	service_inter_check_delay_method=<n/d/s/x.xx>
Example:	service_inter_check_delay_method=s

This option allows you to control how service checks are initially "spread out" in the event queue. Using a "smart" delay calculation (the default) will cause Nagios to calculate an average check interval and spread initial checks of all services out over that interval, thereby helping to eliminate CPU load spikes. Using no delay is generally not recommended, as it will cause all service checks to be scheduled for execution at the same time. This means that you will generally have large CPU spikes when the services are all executed in parallel. More information on how to estimate how the inter-check delay affects service check scheduling can be found *here*. Values are as follows:

- n = Don't use any delay - schedule all service checks to run immediately (i.e. at the same time!)

- d = Use a "dumb" delay of 1 second between service checks
- s = Use a "smart" delay calculation to spread service checks out evenly (default)
- x.xx = Use a user-supplied inter-check delay of x.xx seconds

7.35.9 Inter-Check Sleep Time (Unused)

Format:	sleep_time=<seconds>
Example:	sleep_time=1

This is the number of seconds that Nagios will sleep before checking to see if the next service or host check in the scheduling queue should be executed. Note that Nagios will only sleep after it "catches up" with queued service checks that have fallen behind.

7.35.10 Service Interleave Factor (Unused)

Format:	service_interleave_factor=<s/x>
Example:	service_interleave_factor=s

This variable determines how service checks are interleaved. Interleaving allows for a more even distribution of service checks, reduced load on remote hosts, and faster overall detection of host problems. Setting this value to 1 is equivalent to not interleaving the service checks (this is how versions of Nagios previous to 0.0.5 worked). Set this value to s (smart) for automatic calculation of the interleave factor unless you have a specific reason to change it. The best way to understand how interleaving works is to watch the status CGI (detailed view) when Nagios is just starting. You should see that the service check results are spread out as they begin to appear. More information on how interleaving works can be found *here*.

- x = A number greater than or equal to 1 that specifies the interleave factor to use. An interleave factor of 1 is equivalent to not interleaving the service checks.
- s = Use a "smart" interleave factor calculation (default)

7.35.11 Maximum Concurrent Service Checks (Unused)

Format:	max_concurrent_checks=<max_checks>
Example:	max_concurrent_checks=20

This option allows you to specify the maximum number of service checks that can be run in parallel at any given time. Specifying a value of 1 for this variable essentially prevents any service checks from being run in parallel. Specifying a value of 0 (the default) does not place any restrictions on the number of concurrent checks. You'll have to modify this value based on the system resources you have available on the machine that runs Nagios, as it directly affects the maximum load that will be imposed on the system (processor utilization, memory, etc.). More information on how to estimate how many concurrent checks you should allow can be found *here*.

7.35.12 Check Result Reaper Frequency (Unused)

Format:	check_result_reaper_frequency=<frequency_in_seconds>
Example:	check_result_reaper_frequency=5

This option allows you to control the frequency in seconds of check result "reaper" events. "Reaper" events process the results from host and service checks that have finished executing. These events consitute the core of the monitoring logic in Nagios.

7.35.13 Maximum Check Result Reaper Time

Note: Is it Unused or Not Implemeted??

Format:	max_check_result_reaper_time=<seconds>
Example:	max_check_result_reaper_time=30

This option allows you to control the maximum amount of time in seconds that host and service check result "reaper" events are allowed to run. "Reaper" events process the results from host and service checks that have finished executing. If there are a lot of results to process, reaper events may take a long time to finish, which might delay timely execution of new host and service checks. This variable allows you to limit the amount of time that an individual reaper event will run before it hands control back over to Nagios for other portions of the monitoring logic.

7.35.14 Check Result Path (Unused)

Format:	check_result_path=<path>
Example:	check_result_path=/var/spool/nagios/checkresults

This options determines which directory Nagios will use to temporarily store host and service check results before they are processed. This directory should not be used to store any other files, as Nagios will periodically clean this directory of old file (see the :ref:Max Check Result File Age option above for more information).

Make sure that only a single instance of Nagios has access to the check result path. If multiple instances of Nagios have their check result path set to the same directory, you will run into problems with check results being processed (incorrectly) by the wrong instance of Nagios!

7.35.15 Max Check Result File Age (Unused)

Format:	max_check_result_file_age=<seconds>
Example:	max_check_result_file_age=3600

This options determines the maximum age in seconds that Nagios will consider check result files found in the *check_result_path* directory to be valid. Check result files that are older that this threshold will be deleted by Nagios and the check results they contain will not be processed. By using a value of zero (0) with this option, Nagios will process all check result files - even if they're older than your hardware :-).

7.35.16 Host Inter-Check Delay Method (Unused)

Format:	host_inter_check_delay_method=<n/d/s/x.xx>
Example:	host_inter_check_delay_method=s

This option allows you to control how host checks that are scheduled to be checked on a regular basis are initially "spread out" in the event queue. Using a "smart" delay calculation (the default) will cause Nagios to calculate an average check interval and spread initial checks of all hosts out over that interval, thereby helping to eliminate CPU load spikes. Using no delay is generally not recommended. Using no delay will cause all host checks to be scheduled for execution at the same time. More information on how to estimate how the inter-check delay affects host check scheduling can be found *here*. Values are as follows:

- n = Don't use any delay - schedule all host checks to run immediately (i.e. at the same time!)
- d = Use a "dumb" delay of 1 second between host checks
- s = Use a "smart" delay calculation to spread host checks out evenly (default)
- x.xx = Use a user-supplied inter-check delay of x.xx seconds

7.35.17 Auto-Rescheduling Option (Not implemented)

Format:	auto_reschedule_checks=<0/1>
Example:	auto_reschedule_checks=1

This option determines whether or not Nagios will attempt to automatically reschedule active host and service checks to "smooth" them out over time. This can help to balance the load on the monitoring server, as it will attempt to keep the time between consecutive checks consistent, at the expense of executing checks on a more rigid schedule.

THIS IS AN EXPERIMENTAL FEATURE AND MAY BE REMOVED IN FUTURE VERSIONS. ENABLING THIS OPTION CAN DEGRADE PERFORMANCE - RATHER THAN INCREASE IT - IF USED IMPROPERLY!

7.35.18 Auto-Rescheduling Interval (Not implemented)

Format:	auto_rescheduling_interval=<seconds>
Example:	auto_rescheduling_interval=30

This option determines how often (in seconds) Nagios will attempt to automatically reschedule checks. This option only has an effect if the *Auto-Rescheduling Option* option is enabled. Default is 30 seconds.

THIS IS AN EXPERIMENTAL FEATURE AND MAY BE REMOVED IN FUTURE VERSIONS. ENABLING THE AUTO-RESCHEDULING OPTION CAN DEGRADE PERFORMANCE - RATHER THAN INCREASE IT - IF USED IMPROPERLY!

7.35.19 Auto-Rescheduling Window (Not implemented)

Format:	auto_rescheduling_window=<seconds>
Example:	auto_rescheduling_window=180

This option determines the "window" of time (in seconds) that Nagios will look at when automatically rescheduling checks. Only host and service checks that occur in the next X seconds (determined by this variable) will be rescheduled. This option only has an effect if the Auto-Rescheduling Option option is enabled. Default is 180 seconds (3 minutes).

THIS IS AN EXPERIMENTAL FEATURE AND MAY BE REMOVED IN FUTURE VERSIONS. ENABLING THE AUTO-RESCHEDULING OPTION CAN DEGRADE PERFORMANCE - RATHER THAN INCREASE IT - IF USED IMPROPERLY!

7.35.20 Aggressive Host Checking Option (Unused)

Format:	use_aggressive_host_checking=<0/1>
Example:	use_aggressive_host_checking=0

Nagios tries to be smart about how and when it checks the status of hosts. In general, disabling this option will allow Nagios to make some smarter decisions and check hosts a bit faster. Enabling this option will increase the amount of time required to check hosts, but may improve reliability a bit. Unless you have problems with Nagios not recognizing that a host recovered, I would suggest not enabling this option.

- 0 = Don't use aggressive host checking (default)
- 1 = Use aggressive host checking

7.35.21 Translate Passive Host Checks Option (Not implemented)

Format:	translate_passive_host_checks=<0/1>
Example:	translate_passive_host_checks=1

This option determines whether or not Nagios will translate DOWN/UNREACHABLE passive host check results to their "correct" state from the viewpoint of the local Nagios instance. This can be very useful in distributed and failover monitoring installations. More information on passive check state translation can be found *here*.

- 0 = Disable check translation (default)
- 1 = Enable check translation

7.35.22 Child Process Memory Option (Unused)

Format:	free_child_process_memory=<0/1>
Example:	free_child_process_memory=0

This option determines whether or not Nagios will free memory in child processes when they are fork()ed off from the main process. By default, Nagios frees memory. However, if the *use_large_installation_tweaks* option is enabled, it will not. By defining this option in your configuration file, you are able to override things to get the behavior you want.

- 0 = Don't free memory
- 1 = Free memory

7.35.23 Child Processes Fork Twice (Unused)

Format:	child_processes_fork_twice=<0/1>
Example:	child_processes_fork_twice=0

This option determines whether or not Nagios will fork() child processes twice when it executes host and service checks. By default, Nagios fork()s twice. However, if the *use_large_installation_tweaks* option is enabled, it will only fork() once. By defining this option in your configuration file, you are able to override things to get the behavior you want.

- 0 = Fork() just once
- 1 = Fork() twice

7.35.24 Event Broker Options (Unused)

Format:	event_broker_options=<#>
Example:	event_broker_options=-1

This option controls what (if any) data gets sent to the event broker and, in turn, to any loaded event broker modules. This is an advanced option. When in doubt, either broker nothing (if not using event broker modules) or broker everything (if using event broker modules). Possible values are shown below.

- 0 = Broker nothing
- -1 = Broker everything
- # = See BROKER_* definitions in source code ("include/broker.h") for other values that can be OR'ed together

7.35.25 Event Broker Modules (Unused)

Format:	broker_module=<modulepath> [moduleargs]
Example:	broker_module=/usr/local/nagios/bin/ndomod.o cfg_file=/usr/local/nagios/etc/ndomod.cfg

This directive is used to specify an event broker module that should by loaded by Nagios at startup. Use multiple directives if you want to load more than one module. Arguments that should be passed to the module at startup are seperated from the module path by a space.

Do NOT overwrite modules while they are being used by Nagios or Nagios will crash in a fiery display of SEGFAULT glory. This is a bug/limitation either in "dlopen()", the kernel, and/or the filesystem. And maybe Nagios...

The correct/safe way of updating a module is by using one of these methods:

- Shutdown Nagios, replace the module file, restart Nagios
- While Nagios is running... delete the original module file, move the new module file into place, restart Nagios

7.35.26 Debug File (Unused)

Format:	debug_file=<file_name>
Example:	debug_file=/usr/local/nagios/var/nagios.debug

This option determines where Nagios should write debugging information. What (if any) information is written is determined by the *Debug Level* and *Debug Verbosity* options. You can have Nagios automaticaly rotate the debug file when it reaches a certain size by using the *Maximum Debug File Size* option.

7.35.27 Debug Level (Unused)

Format:	debug_level=<#>
Example:	debug_level=24

This option determines what type of information Nagios should write to the *Debug File*. This value is a logical OR of the values below.

- -1 = Log everything
- 0 = Log nothing (default)
- 1 = Function enter/exit information
- 2 = Config information
- 4 = Process information
- 8 = Scheduled event information
- 16 = Host/service check information
- 32 = Notification information
- 64 = Event broker information

7.35.28 Debug Verbosity (Unused)

Format:	debug_verbosity=<#>
Example:	debug_verbosity=1

This option determines how much debugging information Nagios should write to the *Debug File*.

- 0 = Basic information
- 1 = More detailed information (default)
- 2 = Highly detailed information

7.35.29 Maximum Debug File Size (Unused)

Format:	max_debug_file_size=<#>
Example:	max_debug_file_size=1000000

This option determines the maximum size (in bytes) of the *debug file*. If the file grows larger than this size, it will be renamed with a .old extension. If a file already exists with a .old extension it will automatically be deleted. This helps ensure your disk space usage doesn't get out of control when debugging Nagios.

7.36 Advanced discovery with Shinken

Important: This topic assumes you have read and understood *simple discovery with Shinken*.

7.36.1 How the discovery script works

Did you like the discovery script? Now it's time to look at how it works, and get even more from it.

The discovery is done in two distinct phases:

- the discovery script runs generate raw data
- the discovery rules use this data to generate objects like hosts or services

7.36.2 Discovery scripts

A discovery script can be anything you can launch from a shell, just like plugins. As mentioned their main goal is to generate raw data for objects. Yes, it can be open ports of a server, or the number of wheels your car has, as you want. :)

The raw data is being sent to standard out.

Here is an example of the output of the nmap script for a standard Linux box:

```
$ libexec/nmap_discovery_runner.py -t localhost

localhost::isup=1
localhost::os=linux
localhost::osversion=2.6.X
localhost::macvendor=
localhost::openports=22,80,1521,3306,5432,5666,6502,8080,50000
localhost::fqdn=localhost
localhost::ip=127.0.0.1
```

So the output format is:

```
objectname::key=value
```

If there are multiple values, like here for open ports, they are separated by commas "'‘,’'".

The discovery script definitions (like nmap or vmware used by default) are located in the file ‘’/etc/shinken/discovery_runs.cfg’‘.

7.36.3 Discovery rules

Without rules, the raw data that is being generated by the discovery scripts is useless. The rules are defined in the ‘’/etc/shinken/discovery_rules.cfg" file.

Host rule

Here is an example of how to create a "generic" host for anything that is detected by nmap and answers to a ping request:

```
define discoveryrule {
       discoveryrule_name       HostGeneric
       creation_type            host

       isup                     1

       use                      generic-host
}
```

There are three main parts for a rule:

- ‘’discoveryrule_name" and ‘’creation_type" parameter. The first one should be unique, and the second can be ‘host’ or ‘service’ (default). More types will be added in the future.
- ‘’isup’‘: refers the key name that will be looked up in the raw data from the discovery scripts. It's value (here 1) will be used for a comparison. If all key/values pairs are good, the rule is valid, and will be applied.
- ‘’use’‘: This mentions the template from which the generated object will inherit from. You can add as many properties as you want.

Service rule

Here is an example for a port matching rule service creation:

```
define discoveryrule {
     discoveryrule_name       Ftp

     openports                ^21$

     check_command            check_ftp
     service_description      Ftp
     use                      generic-service
}
```

Here, if the port 21 is open. The ^and $ is for the regexp thing, so 21 and only 21 will be match, and not 210 for example.

The service generated will be with FTP for the host_name the object_name send by the discovery script, the check_command check_ftp and it will use the generic-service template.

The ! (not) key

You can ask **not** to match a rule. It's very easy, just add a ! character before the key name.

For example:

```
define discoveryrule {
     discoveryrule_name       Ftp

     openports                ^21$
     !os                      linux

     check_command            check_ftp
     service_description      Ftp
     use                      generic-service
}
```

This will create the Ftp service for all hosts that have port 21 open, but not for the linux ones.

Add something instead of replace

By default, when you put a new host/service property, it will replace all previously detected values. For some properties like templates or groups, this is not a good idea. That's why you can say a property should be "added" by using the character "+" before it.

For example, we want to add the "ftp" and "http" templates on the host, without removing all previously inserted values.

```
define discoveryrule {
     discoveryrule_name       Ftp
     creation_type            host
     openports                ^21$
     +use                     ftp
}

define discoveryrule {
     discoveryrule_name       Http
     creation_type            host
     openports                ^21$
     +use                     http
}
```

If both ports are open, it will create an host with:

```
define host {
  host_name   localhost
  use         ftp,http
}
```

Important: The rules order is important, here ftp apply before http. So put the "generic" template at the end of you rules file.

Important: Why is the rule order important, explain the impact.

Delete something after add

Sometimes you need to simply remove a property that conflicts with a new one. For example, some routers are derived from linux system but does not work with the linux template. That's why you can say a property should be "remove" by using the character "-" before it.

For exemple we want to add the "router-os" template but not the "linux" template on the host and do not remove previously inserted values.

```
define discoveryrule {
     discoveryrule_name       Ftp
     creation_type            host
     openports                ^21$
     +use                     ftp
}

define discoveryrule {
     discoveryrule_name       Http
     creation_type            host
     openports                ^21$
     +use                     http
}

define discoveryrule {
     discoveryrule_name       Linux
     creation_type            host
     os                       linux
     +use                     linux
}

define discoveryrule {
     discoveryrule_name       RouterOS
     creation_type            host
     macvendor                routerboard
     +use                     router-os
     -use                     linux
}
```

If both ports are open, os detected is linux and the macvendor is routerboard it will create an host with:

```
define host {
  host_name   myrouter
  use         ftp,http,router-os
}
```

7.37 Discovery with Shinken

7.37.1 Simple use of the discovery tool

When Shinken is installed, the discovery script shinken-discovery can help you start your new monitoring tool and integrate a large number of hosts. This does not not replace extracting data from an authoritative CMDB/IT reference for provisioning known hosts. It can be used to supplement the data from the authoritative references.

At this time, two "discovery" modules are available:

- Network based discovery using nmap

- VMware based discovery, using the check_esx3.pl script communicating with a vCenter installation.

It is suggested to execute both discovery modules in one pass, because one module can use data from the other.

Setup nmap discovery

The network discovery scans your network and sets up a basic monitoring configuration for all your hosts and network services. It uses the nmap tool.

Ubuntu:

```
sudo apt-get install nmap
```

RedHat/Centos:

```
yum install nmap
```

Windows: Not available at this time.

You need to setup the nmap targets in the file /usr/local/shinken/etc/resource.cfg: For nmap:

```
$NMAPTARGETS$=localhost www.google.fr 192.168.0.1-254
```

This will scan the localhost, one of the numerous Google server and your LAN. Change it to your own LAN values of course!

Tip: This value can be changed without modifying this file with the -m discovery script argument

Setup the VMware part

Tip: Of course, if you do not have a vCenter installation, skip this part ...

You will need the check_esx3.pl script. You can get it at http://www.op5.org/community/plugin-inventory/op5-projects/op5-plugins and install it in your standard plugin directory (should be /var/lib/plugins/nagios by default).

You need to setup vcenter acces in the file /etc/shinken/resource.cfg: Enter your server and credential (can be an account domain)

```
$VCENTER$=vcenter.mydomain.com
$VCENTERLOGIN$=someuser
$VCENTERPASSWORD$=somepassowrd
```

Launch it!

Now, you are ready to run the discovery tool:

This call will create hosts and services for nmap and vmware (vsphere) scripts in the /etc/shinken/object/discovery directory.

```
sudo shinken-discovery -o /etc/shinken/objects/discovery -r nmap,vsphere
```

If you are lazy and do not want to edit the resource file, you can set macros with the -m arguments:

```
sudo shinken-discovery -o /etc/shinken/objects/discovery -r nmap -m "NMAPTARGETS=192.168.0.1-254 loca
```

You can set several macros, just put them on the same -m argument, separated by a comma (,).

Tip: The scan can take quite a few minutes if you are scanning a large network, you can go have a coffee. The scan timeout is set to 60 minutes.

Restart Shinken

Once the scan is completed, you can restart Shinken and enjoy your new hosts and services:

```
sudo /etc/init.d/shinken restart
```

7.37.2 More about discovery

If you want to know more about the discovery process, like how to create a discovery script or define creation rules, consult the *advanced discovery* documentation.

CHAPTER 8

Config

8.1 Host Definition

8.1.1 Description

A host definition is used to define a physical server, workstation, device, etc. that resides on your network.

8.1.2 Definition Format

Bold directives are required, while the others are optional.

define host{	
host_name	***host_name***
alias	***alias***
display_name	*display_name*
address	***address***
parents	*host_names*
hostgroups	*hostgroup_names*
check_command	*command_name*
initial_state	[o,d,u]
max_check_attempts	**#**
check_interval	#
retry_interval	#
active_checks_enabled	[0/1]
passive_checks_enabled	[0/1]
check_period	***timeperiod_name***
obsess_over_host	[0/1]
check_freshness	[0/1]
freshness_threshold	#
event_handler	*command_name*
event_handler_enabled	[0/1]
low_flap_threshold	#
high_flap_threshold	#
flap_detection_enabled	[0/1]
flap_detection_options	[o,d,u]
process_perf_data	[0/1]
retain_status_information	[0/1]
retain_nonstatus_information	[0/1]
contacts	***contacts***
contact_groups	***contact_groups***
notification_interval	**#**
first_notification_delay	#
notification_period	***timeperiod_name***
notification_options	[d,u,r,f,s]
notifications_enabled	[0/1]
stalking_options	[o,d,u]
notes	*note_string*
notes_url	*url*
action_url	*url*
icon_image	*image_file*
icon_image_alt	*alt_string*

Continued on next page

Table 8.1 – continued from previous page

vrml_image	*image_file*
statusmap_image	*image_file*
2d_coords	*x_coord,y_coord*
3d_coords	*x_coord,y_coord,z_coord*
realm	*realm*
poller_tag	*poller_tag*
business_impact	[0/1/2/3/4/5]
resultmodulations	*resultmodulations*
escalations	*escalations names*
business_impact_modulations	*business_impact_modulations names*
icon_set	[database/disk/network_service/server]
maintenance_period	*timeperiod_name*
}	

8.1.3 Example Definition

```
define host{
host_name                       bogus-router
alias                           Bogus Router #1
address                         192.168.1.254
parents                         server-backbone
check_command                   check-host-alive
check_interval                  5
retry_interval                  1
max_check_attempts              5
check_period                    24x7
process_perf_data               0
retain_nonstatus_information    0
contact_groups                  router-admins
notification_interval           30
notification_period             24x7
notification_options            d,u,r
realm                           Europe
poller_tag                      DMZ
icon_set                        server
}
```

8.1.4 Directive Descriptions

host_name

This directive is used to define a short name used to identify the host. It is used in host group and service definitions to reference this particular host. Hosts can have multiple services (which are monitored) associated with them. When used properly, the $HOSTNAME$ *macro* will contain this short name.

alias

This directive is used to define a longer name or description used to identify the host. It is provided in order to allow you to more easily identify a particular host. When used properly, the $HOSTALIAS$ *macro* will contain this alias/description.

address

This directive is used to define the address of the host. Normally, this is an IP address, although it could really be anything you want (so long as it can be used to check the status of the host). You can use a FQDN to identify the host instead of an IP address, but if "DNS" services are not available this could cause problems. When used properly, the $HOSTADDRESS$ *macro* will contain this address.

If you do not specify an address directive in a host definition, the name of the host will be used as its address.

A word of caution about doing this, however - if "DNS" fails, most of your service checks will fail because the plugins will be unable to resolve the host name.

display_name

This directive is used to define an alternate name that should be displayed in the web interface for this host. If not specified, this defaults to the value you specify for the *host_name* directive.

parents

This directive is used to define a comma-delimited list of short names of the "parent" hosts for this particular host. Parent hosts are typically routers, switches, firewalls, etc. that lie between the monitoring host and a remote hosts. A router, switch, etc. which is closest to the remote host is considered to be that host's "parent". Read the "Determining Status and Reachability of Network Hosts" document located *here* for more information. If this host is on the same network segment as the host doing the monitoring (without any intermediate routers, etc.) the host is considered to be on the local network and will not have a parent host. Leave this value blank if the host does not have a parent host (i.e. it is on the same segment as the Shinken host). The order in which you specify parent hosts has no effect on how things are monitored.

hostgroups

This directive is used to identify the *short name(s)* of the *hostgroup(s)* that the host belongs to. Multiple hostgroups should be separated by commas. This directive may be used as an alternative to (or in addition to) using the *members* directive in *hostgroup* definitions.

check_command

This directive is used to specify the *short name* of the *Command Definition* that should be used to check if the host is up or down. Typically, this command would try and ping the host to see if it is "alive". The command must return a status of OK (0) or Shinken will assume the host is down. If you leave this argument blank, the host will *not* be actively checked. Thus, Shinken will likely always assume the host is up (it may show up as being in a "PENDING" state in the web interface). This is useful if you are monitoring printers or other devices that are frequently turned off. The maximum amount of time that the notification command can run is controlled by the *host_check_timeout* option.

initial_state

By default Shinken will assume that all hosts are in UP states when in starts. You can override the initial state for a host by using this directive. Valid options are: **o** = UP, **d** = DOWN, and **u** = UNREACHABLE.

max_check_attempts

This directive is used to define the number of times that Shinken will retry the host check command if it returns any state other than an OK state. Setting this value to 1 will cause Shinken to generate an alert without retrying the host check again.

If you do not want to check the status of the host, you must still set this to a minimum value of 1. To bypass the host check, just leave the "check_command" option blank.

check_interval

This directive is used to define the number of "time units" between regularly scheduled checks of the host. Unless you've changed the *interval_length* directive from the default value of 60, this number will mean minutes. More information on this value can be found in the *check scheduling* documentation.

retry_interval

This directive is used to define the number of "time units" to wait before scheduling a re-check of the hosts. Hosts are rescheduled at the retry interval when they have changed to a non-UP state. Once the host has been retried **max_check_attempts** times without a change in its status, it will revert to being scheduled at its "normal" rate as defined by the **check_interval** value. Unless you've changed the *interval_length* directive from the default value of 60, this number will mean minutes. More information on this value can be found in the *check cheduling* documentation.

active_checks_enabled

This directive is used to determine whether or not active checks (either regularly scheduled or on-demand) of this host are enabled. Values: 0 = disable active host checks, 1 = enable active host checks.

passive_checks_enabled

This directive is used to determine whether or not passive checks are enabled for this host. Values: 0 = disable passive host checks, 1 = enable passive host checks.

check_period

This directive is used to specify the short name of the *time period* during which active checks of this host can be made.

obsess_over_host

This directive determines whether or not checks for the host will be "obsessed" over using the *ochp_command*.

check_freshness *

This directive is used to determine whether or not *freshness checks* are enabled for this host. Values: 0 = disable freshness checks, 1 = enable freshness checks.

freshness_threshold

This directive is used to specify the freshness threshold (in seconds) for this host. If you set this directive to a value of 0, Shinken will determine a freshness threshold to use automatically.

event_handler

This directive is used to specify the *short name* of the *command* that should be run whenever a change in the state of the host is detected (i.e. whenever it goes down or recovers). Read the documentation on *event handlers* for a more detailed explanation of how to write scripts for handling events. The maximum amount of time that the event handler command can run is controlled by the *event_handler_timeout* option.

event_handler_enabled

This directive is used to determine whether or not the event handler for this host is enabled. Values: 0 = disable host event handler, 1 = enable host event handler.

low_flap_threshold

This directive is used to specify the low state change threshold used in flap detection for this host. More information on flap detection can be found *here*. If you set this directive to a value of 0, the program-wide value specified by the *low_host_flap_threshold* directive will be used.

high_flap_threshold

This directive is used to specify the high state change threshold used in flap detection for this host. More information on flap detection can be found *here*. If you set this directive to a value of 0, the program-wide value specified by the *high_host_flap_threshold* directive will be used.

flap_detection_enabled

This directive is used to determine whether or not flap detection is enabled for this host. More information on flap detection can be found *here*. Values: 0 = disable host flap detection, 1 = enable host flap detection.

flap_detection_options

This directive is used to determine what host states the *flap detection logic* will use for this host. Valid options are a combination of one or more of the following: **o** = UP states, **d** = DOWN states, **u** = UNREACHABLE states.

process_perf_data

This directive is used to determine whether or not the processing of performance data is enabled for this host. Values: 0 = disable performance data processing, 1 = enable performance data processing.

retain_status_information

This directive is used to determine whether or not status-related information about the host is retained across program restarts. This is only useful if you have enabled state retention using the *retain_state_information* directive. Value: 0 = disable status information retention, 1 = enable status information retention.

retain_nonstatus_information

This directive is used to determine whether or not non-status information about the host is retained across program restarts. This is only useful if you have enabled state retention using the *retain_state_information* directive. Value: 0 = disable non-status information retention, 1 = enable non-status information retention.

contacts

This is a list of the *short names* of the *contacts* that should be notified whenever there are problems (or recoveries) with this host. Multiple contacts should be separated by commas. Useful if you want notifications to go to just a few people and don't want to configure *contact groups*. You must specify at least one contact or contact group in each host definition.

contact_groups

This is a list of the *short names* of the *contact groups* that should be notified whenever there are problems (or recoveries) with this host. Multiple contact groups should be separated by commas. You must specify at least one contact or contact group in each host definition.

notification_interval

This directive is used to define the number of "time units" to wait before re-notifying a contact that this service is *still* down or unreachable. Unless you've changed the *interval_length* directive from the default value of 60, this number will mean minutes. If you set this value to 0, Shinken will *not* re-notify contacts about problems for this host - only one problem notification will be sent out.

first_notification_delay

This directive is used to define the number of "time units" to wait before sending out the first problem notification when this host enters a non-UP state. Unless you've changed the *interval_length* directive from the default value of 60, this number will mean minutes. If you set this value to 0, Shinken will start sending out notifications immediately.

notification_period

This directive is used to specify the short name of the *time period* during which notifications of events for this host can be sent out to contacts. If a host goes down, becomes unreachable, or recoveries during a time which is not covered by the time period, no notifications will be sent out.

notification_options

This directive is used to determine when notifications for the host should be sent out. Valid options are a combination of one or more of the following: **d** = send notifications on a DOWN state, **u** = send notifications on an UNREACHABLE state, **r** = send notifications on recoveries (OK state), **f** = send notifications when the host starts and stops *flapping*, and **s** = send notifications when *scheduled downtime* starts and ends. If you specify **n** (none) as an option, no host notifications will be sent out. If you do not specify any notification options, Shinken will assume that you want notifications to be sent out for all possible states.

If you specify **d,r** in this field, notifications will only be sent out when the host goes DOWN and when it recovers from a DOWN state.

notifications_enabled

This directive is used to determine whether or not notifications for this host are enabled. Values: 0 = disable host notifications, 1 = enable host notifications.

stalking_options

This directive determines which host states "stalking" is enabled for. Valid options are a combination of one or more of the following: **o** = stalk on UP states, **d** = stalk on DOWN states, and **u** = stalk on UNREACHABLE states. More information on state stalking can be found *here*.

notes

This directive is used to define an optional string of notes pertaining to the host. If you specify a note here, you will see the it in the extended information CGI (when you are viewing information about the specified host).

notes_url

This variable is used to define an optional URL that can be used to provide more information about the host. If you specify an URL, you will see a red folder icon in the CGIs (when you are viewing host information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will the the same as what is used to access the CGIs (i.e. ///cgi-bin/shinken///). This can be very useful if you want to make detailed information on the host, emergency contact methods, etc. available to other support staff.

action_url

This directive is used to define one or more optional URL that can be used to provide more actions to be performed on the host. If you specify an URL, you will see a red "splat" icon in the CGIs (when you are viewing host information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will the the same as what is used to access the CGIs (i.e. */cgi-bin/shinken/*). *Configure multiple action_urls.*

icon_image

This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be displayed in the various places in the CGIs. The image will look best if it is 40x40 pixels in size. Images for hosts are assumed to be in the **logos/** subdirectory in your HTML images directory (i.e. "/usr/local/shinken/share/images/logos").

icon_image_alt

This variable is used to define an optional string that is used in the ALT tag of the image specified by the *<icon_image>* argument.

vrml_image

This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be used as the texture map for the specified host in the statuswrl CGI. Unlike the image you use for the *<icon_image>* variable, this one should probably *not* have any transparency. If it does, the host object will look a bit wierd. Images for hosts are assumed to be in the **logos/** subdirectory in your HTML images directory (i.e. "/usr/local/shinken/share/images/logos").

statusmap_image

This variable is used to define the name of an image that should be associated with this host in the statusmap CGI. You can specify a JPEG, PNG, and GIF image if you want, although I would strongly suggest using a GD2 format image, as other image formats will result in a lot of wasted CPU time when the statusmap image is generated. GD2 images can be created from PNG images by using the **pngtogd2** utility supplied with Thomas Boutell's gd library. The GD2 images should be created in *uncompressed* format in order to minimize CPU load when the statusmap CGI is generating the network map image. The image will look best if it is 40x40 pixels in size. You can leave these option blank if you are not using the statusmap CGI. Images for hosts are assumed to be in the **logos/** subdirectory in your HTML images directory (i.e. "/usr/local/shinken/share/images/logos").

2d_coords

This variable is used to define coordinates to use when drawing the host in the statusmap CGI. Coordinates should be given in positive integers, as they correspond to physical pixels in the generated image. The origin for drawing (0,0) is in the upper left hand corner of the image and extends in the positive x direction (to the right) along the top of the image and in the positive y direction (down) along the left hand side of the image. For reference, the size of the icons drawn is usually about 40x40 pixels (text takes a little extra space). The coordinates you specify here are for the upper left hand corner of the host icon that is drawn.

Don't worry about what the maximum x and y coordinates that you can use are. The CGI will automatically calculate the maximum dimensions of the image it creates based on the largest x and y coordinates you specify.

3d_coords

This variable is used to define coordinates to use when drawing the host in the statuswrl CGI. Coordinates can be positive or negative real numbers. The origin for drawing is (0.0,0.0,0.0). For reference, the size of the host cubes drawn is 0.5 units on each side (text takes a little more space). The coordinates you specify here are used as the center of the host cube.

realm

This variable is used to define the *realm* where the host will be put. By putting the host in a realm, it will be manage by one of the scheduler of this realm.

poller_tag

This variable is used to define the poller_tag of the host. All checks of this hosts will only take by pollers that have this value in their poller_tags parameter.

By default the pollerag value is 'None', so all untagged pollers can take it because None is set by default for them.

business_impact

This variable is used to set the importance we gave to this host for the business from the less important (0 = nearly nobody will see if it's in error) to the maximum (5 = you lost your job if it fail). The default value is 2.

resultmodulations

This variable is used to link with resultmodulations objects. It will allow such modulation to apply, like change a warning in critical for this host.

escalations

This variable is used to link with escalations objects. It will allow such escalations rules to appy. Look at escalations objects for more details.

business_impact_modulations

This variable is used to link with business_impact_modulations objects. It will allow such modulation to apply (for example if the host is a payd server, it will be important only in a specific timeperiod: near the payd day). Look at business_impact_modulations objects for more details.

icon_set

This variable is used to set the icon in the Shinken Webui. For now, values are only : database, disk, network_service, server

maintenance_period

Shinken-specific variable to specify a recurring downtime period. This works like a scheduled downtime, so unlike a check_period with exclusions, checks will still be made (no "*blackout*" times). announcement

8.2 Host Group Definition

8.2.1 Description

A host group definition is used to group one or more hosts together for simplifying configuration with *object tricks* or display purposes in the CGIs .

8.2.2 Definition Format

Bold directives are required, while the others are optional.

define hostgroup{	
hostgroup_name	***hostgroup_name***
alias	***alias***
members	*hosts*
hostgroup_members	*hostgroups*
notes	*note_string*
notes_url	*url*
action_url	*url*
realm	*realm*
}	

8.2.3 Example Definition

```
define hostgroup{
hostgroup_name          novell-servers
alias                   Novell Servers
members                 netware1,netware2,netware3,netware4
realm                   Europe
}
```

8.2.4 Directive Descriptions

hostgroup_name

This directive is used to define a short name used to identify the host group.

alias

This directive is used to define is a longer name or description used to identify the host group. It is provided in order to allow you to more easily identify a particular host group.

members

This is a list of the *short names* of *hosts* that should be included in this group. Multiple host names should be separated by commas. This directive may be used as an alternative to (or in addition to) the *hostgroups* directive in *host definitions*.

hostgroup_members

This optional directive can be used to include hosts from other "sub" host groups in this host group. Specify a comma-delimited list of short names of other host groups whose members should be included in this group.

notes

This directive is used to define an optional string of notes pertaining to the host. If you specify a note here, you will see the it in the extended information CGI (when you are viewing information about the specified host).

notes_url

This variable is used to define an optional URL that can be used to provide more information about the host group. If you specify an URL, you will see a red folder icon in the CGIs (when you are viewing hostgroup information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will the the same as what is used to access the CGIs (i.e. ///cgi-bin/nagios///). This can be very useful if you want to make detailed information on the host group, emergency contact methods, etc. available to other support staff.

action_url

This directive is used to define an optional URL that can be used to provide more actions to be performed on the host group. If you specify an URL, you will see a red "splat" icon in the CGIs (when you are viewing hostgroup information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will the the same as what is used to access the CGIs (i.e. ///cgi-bin/shinken///).

realm

This directive is used to define in which *realm* all hosts of this hostgroup will be put into. If the host are already tagged by a realm (and not the same), the value taken into account will the the one of the host (and a warning will be raised). If no realm is defined, the default one will be take..

8.3 Service Definition

8.3.1 Description

A service definition is used to identify a "service" that runs on a host. The term "service" is used very loosely. It can mean an actual service that runs on the host (POP, "SMTP", "HTTP", etc.) or some other type of metric associated with the host (response to a ping, number of logged in users, free disk space, etc.). The different arguments to a service definition are outlined below.

8.3.2 Definition Format

Bold directives are required, while the others are optional.

define service{	
host_name	***host_name***
hostgroup_name	*hostgroup_name*
service_description	***service_description***
display_name	*display_name*
servicegroups	servicegroup_names
is_volatile	[0/1]
check_command	***command_name***
initial_state	[o,w,u,c]
max_check_attempts	**#**
check_interval	**#**
retry_interval	**#**
active_checks_enabled	[0/1]
passive_checks_enabled	[0/1]
check_period	***timeperiod_name***

Continued on next page

Table 8.2 – continued from previous page

obsess_over_service	[0/1]
check_freshness	[0/1]
freshness_threshold	#
event_handler	*command_name*
event_handler_enabled	[0/1]
low_flap_threshold	#
high_flap_threshold	#
flap_detection_enabled	[0/1]
flap_detection_options	[o,w,c,u]
process_perf_data	[0/1]
retain_status_information	[0/1]
retain_nonstatus_information	[0/1]
notification_interval	**#**
first_notification_delay	#
notification_period	***timeperiod_name***
notification_options	[w,u,c,r,f,s]
notifications_enabled	[0/1]
contacts	***contacts***
contact_groups	***contact_groups***
stalking_options	[o,w,u,c]
notes	*note_string*
notes_url	*url*
action_url	*url*
icon_image	*image_file*
icon_image_alt	*alt_string*
poller_tag	*poller_tag*
service_dependencies	*host,service_description*
business_impact	[0/1/2/3/4/5]
icon_set	[database/disk/network_service/server]
maintenance_period	*timeperiod_name*
}	

8.3.3 Example Definition

```
define service{
host_name               linux-server
service_description     check-disk-sda1
check_command           check-disk!/dev/sda1
max_check_attempts      5
check_interval          5
retry_interval          3
check_period            24x7
notification_interval   30
notification_period     24x7
notification_options    w,c,r
contact_groups          linux-admins
poller_tag              DMZ
icon_set                server
}
```

8.3.4 Directive Descriptions:

host_name

This directive is used to specify the *short name(s)* of the *host(s)* that the service "runs" on or is associated with. Multiple hosts should be separated by commas.

hostgroup_name

This directive is used to specify the *short name(s)* of the *hostgroup(s)* that the service "runs" on or is associated with. Multiple hostgroups should be separated by commas. The hostgroup_name may be used instead of, or in addition to, the host_name directive.

This is possibleto define "complex" hostgroup expression with the folowing operators :

- & : it's use to make an AND betweens groups
- : it's use to make an OR betweens groups
- ! : it's use to make a NOT of a group or expression
- , : it's use to make a OR, like the | sign.
- (and) : they are use like in all math expressions.

For example the above definition is valid

```
hostgroup_name=(linux|windows)&!qualification,routers
```

This service wil be apply on hosts that are in the routers group or (in linux or windows and not in qualification group).

service_description

This directive is used to define the description of the service, which may contain spaces, dashes, and colons (semicolons, apostrophes, and quotation marks should be avoided). No two services associated with the same host can have the same description. Services are uniquely identified with their *host_name* and *service_description* directives.

display_name

This directive is used to define an alternate name that should be displayed in the web interface for this service. If not specified, this defaults to the value you specify for the *service_description* directive.

The current CGIs do not use this option, although future versions of the web interface will.

servicegroups

This directive is used to identify the *short name(s)* of the *servicegroup(s)* that the service belongs to. Multiple servicegroups should be separated by commas. This directive may be used as an alternative to using the *members* directive in *servicegroup* definitions.

is_volatile

This directive is used to denote whether the service is "volatile". Services are normally *not* volatile. More information on volatile service and how they differ from normal services can be found *here*. Value: 0 = service is not volatile, 1 = service is volatile.

check_command

This directive is used to specify the *short name* of the *command* that Shinken will run in order to check the status of the service. The maximum amount of time that the service check command can run is controlled by the *service_check_timeout* option. There is also a command with the reserved name "bp_rule". It is defined internally and has a special meaning. Unlike other commands it mustn't be registered in a command definition. It's purpose is not to execute a plugin but to represent a logical operation on the statuses of other services. It is possible to define logical relationships with the following operators :

- & : it's use to make an AND betweens statuses
- : it's use to make an OR betweens statuses
- ! : it's use to make a NOT of a status or expression
- , : it's use to make a OR, like the | sign.
- (and) : they are used like in all math expressions.

For example the following definition of a business process rule is valid

```
bp_rule!(websrv1,apache | websrv2,apache) & dbsrv1,oracle
```

If at least one of the apaches on servers websrv1 and websrv2 is OK and if the oracle database on dbsrv1 is OK then the rule and thus the service is OK

initial_state

By default Shinken will assume that all services are in OK states when in starts. You can override the initial state for a service by using this directive. Valid options are:

- **o** = OK
- **w** = WARNING
- **u** = UNKNOWN
- **c** = CRITICAL.

max_check_attempts

This directive is used to define the number of times that Shinken will retry the service check command if it returns any state other than an OK state. Setting this value to 1 will cause Shinken to generate an alert without retrying the service check again.

check_interval

This directive is used to define the number of "time units" to wait before scheduling the next "regular" check of the service. "Regular" checks are those that occur when the service is in an OK state or when the service is in a non-OK state, but has already been rechecked **max_check_attempts** number of times. Unless you've changed the *interval_length* directive from the default value of 60, this number will mean minutes. More information on this value can be found in the *check scheduling* documentation.

retry_interval

This directive is used to define the number of "time units" to wait before scheduling a re-check of the service. Services are rescheduled at the retry interval when they have changed to a non-OK state. Once the service has been retried **max_check_attempts** times without a change in its status, it will revert to being scheduled at its "normal" rate as defined by the **check_interval** value. Unless you've changed the *interval_length* directive from the default value of 60, this number will mean minutes. More information on this value can be found in the *check scheduling* documentation.

active_checks_enabled *

This directive is used to determine whether or not active checks of this service are enabled. Values:

- 0 = disable active service checks
- 1 = enable active service checks.

passive_checks_enabled *

This directive is used to determine whether or not passive checks of this service are enabled. Values:

- 0 = disable passive service checks
- 1 = enable passive service checks.

check_period

This directive is used to specify the short name of the *time period* during which active checks of this service can be made.

obsess_over_service *

This directive determines whether or not checks for the service will be "obsessed" over using the *ocsp_command*.

check_freshness *

This directive is used to determine whether or not *freshness checks* are enabled for this service. Values:

- 0 = disable freshness checks
- 1 = enable freshness checks

freshness_threshold

This directive is used to specify the freshness threshold (in seconds) for this service. If you set this directive to a value of 0, Shinken will determine a freshness threshold to use automatically.

event_handler

This directive is used to specify the *short name* of the *Command Definition* that should be run whenever a change in the state of the service is detected (i.e. whenever it goes down or recovers). Read the documentation on *event handlers* for a more detailed explanation of how to write scripts for handling events. The maximum amount of time that the event handler command can run is controlled by the *event_handler_timeout* option.

event_handler_enabled *

This directive is used to determine whether or not the event handler for this service is enabled. Values:

- 0 = disable service event handler
- 1 = enable service event handler.

low_flap_threshold

This directive is used to specify the low state change threshold used in flap detection for this service. More information on flap detection can be found *here*. If you set this directive to a value of 0, the program-wide value specified by the *low_service_flap_threshold* directive will be used.

high_flap_threshold

This directive is used to specify the high state change threshold used in flap detection for this service. More information on flap detection can be found *here*. If you set this directive to a value of 0, the program-wide value specified by the *high_service_flap_threshold* directive will be used.

flap_detection_enabled *

This directive is used to determine whether or not flap detection is enabled for this service. More information on flap detection can be found *here*. Values:

- 0 = disable service flap detection
- 1 = enable service flap detection.

flap_detection_options

This directive is used to determine what service states the *flap detection logic* will use for this service. Valid options are a combination of one or more of the following :

- **o** = OK states
- **w** = WARNING states
- **c** = CRITICAL states

- **u** = UNKNOWN states.

process_perf_data *

This directive is used to determine whether or not the processing of performance data is enabled for this service. Values:

- 0 = disable performance data processing
- 1 = enable performance data processing

retain_status_information

This directive is used to determine whether or not status-related information about the service is retained across program restarts. This is only useful if you have enabled state retention using the *retain_state_information* directive. Value:

- 0 = disable status information retention
- 1 = enable status information retention.

retain_nonstatus_information

This directive is used to determine whether or not non-status information about the service is retained across program restarts. This is only useful if you have enabled state retention using the *retain_state_information* directive. Value:

- 0 = disable non-status information retention
- 1 = enable non-status information retention

notification_interval

This directive is used to define the number of "time units" to wait before re-notifying a contact that this service is *still* in a non-OK state. Unless you've changed the *interval_length* directive from the default value of 60, this number will mean minutes. If you set this value to 0, Shinken will *not* re-notify contacts about problems for this service - only one problem notification will be sent out.

first_notification_delay

This directive is used to define the number of "time units" to wait before sending out the first problem notification when this service enters a non-OK state. Unless you've changed the *interval_length* directive from the default value of 60, this number will mean minutes. If you set this value to 0, Shinken will start sending out notifications immediately.

notification_period

This directive is used to specify the short name of the *time period* during which notifications of events for this service can be sent out to contacts. No service notifications will be sent out during times which is not covered by the time period.

notification_options

This directive is used to determine when notifications for the service should be sent out. Valid options are a combination of one or more of the following:

- **w** = send notifications on a WARNING state
- **u** = send notifications on an UNKNOWN state
- **c** = send notifications on a CRITICAL state
- **r** = send notifications on recoveries (OK state)
- **f** = send notifications when the service starts and stops *flapping*
- **s** = send notifications when *scheduled downtime* starts and ends

- **n** (none) as an option, no service notifications will be sent out. If you do not specify any notification options, Shinken will assume that you want notifications to be sent out for all possible states

If you specify **w,r** in this field, notifications will only be sent out when the service goes into a WARNING state and when it recovers from a WARNING state.

notifications_enabled *

This directive is used to determine whether or not notifications for this service are enabled. Values:

- 0 = disable service notifications
- 1 = enable service notifications.

contacts

This is a list of the *short names* of the *contacts* that should be notified whenever there are problems (or recoveries) with this service. Multiple contacts should be separated by commas. Useful if you want notifications to go to just a few people and don't want to configure *contact groups*. You must specify at least one contact or contact group in each service definition.

contact_groups

This is a list of the *short names* of the *contact groups* that should be notified whenever there are problems (or recoveries) with this service. Multiple contact groups should be separated by commas. You must specify at least one contact or contact group in each service definition.

stalking_options

This directive determines which service states "stalking" is enabled for. Valid options are a combination of one or more of the following :

- o = stalk on OK states
- w = stalk on WARNING states
- u = stalk on UNKNOWN states
- c = stalk on CRITICAL states

More information on state stalking can be found *here*.

notes

This directive is used to define an optional string of notes pertaining to the service. If you specify a note here, you will see the it in the extended information CGI (when you are viewing information about the specified service).

notes_url

This directive is used to define an optional URL that can be used to provide more information about the service. If you specify an URL, you will see a red folder icon in the CGIs (when you are viewing service information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will the the same as what is used to access the CGIs (i.e. ///cgi-bin/shinken///). This can be very useful if you want to make detailed information on the service, emergency contact methods, etc. available to other support staff.

action_url

This directive is used to define an optional URL that can be used to provide more actions to be performed on the service. If you specify an URL, you will see a red "splat" icon in the CGIs (when you are viewing service information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will the the same as what is used to access the CGIs (i.e. ///cgi-bin/shinken///).

icon_image

This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this service. This image will be displayed in the status and extended information CGIs. The image will look best if it is 40x40 pixels in size. Images for services are assumed to be in the **logos/** subdirectory in your HTML images directory (i.e. "/usr/local/shinken/share/images/logos").

icon_image_alt

This variable is used to define an optional string that is used in the ALT tag of the image specified by the *<icon_image>* argument. The ALT tag is used in the status, extended information and statusmap CGIs.

poller_tag

This variable is used to define the poller_tag of checks from this service. All of theses checks will be taken by pollers that have this value in their poller_tags parameter.

By default there is no poller_tag, so all untaggued pollers can take it.

service_dependencies

This variable is used to define services that this service is dependent of for notifications. It's a comma separated list of services: host,service_description,host,service_description. For each service a service_dependency will be created with default values (notification_failure_criteria as 'u,c,w' and no dependency_period). For more complex failure criteria or dpendency period you must create a service_dependency object, as described in *advanced dependency configuraton*. The host can be omitted from the configuration, which means that the service dependency is for the same host.

```
service_dependencies    hostA,service_descriptionA,hostB,service_descriptionB
service_dependencies    ,service_descriptionA,,service_descriptionB,hostC,service_descriptionC
```

By default this value is void so there is no linked dependencies. This is typically used to make a service dependant on an agent software, like an NRPE check dependant on the availability of the NRPE agent.

business_impact

This variable is used to set the importance we gave to this service from the less important (0 = nearly nobody will see if it's in error) to the maximum (5 = you lost your job if it fail). The default value is 2.

icon_set

This variable is used to set the icon in the Shinken Webui. For now, values are only : database, disk, network_service, server

maintenance_period

Shinken-specific variable to specify a recurring downtime period. This works like a scheduled downtime, so unlike a check_period with exclusions, checks will still be made (no "blackout" times). announcement

8.4 Service Group Definition

8.4.1 Description

A service group definition is used to group one or more services together for simplifying configuration with *object tricks* or display purposes in the CGIs.

8.4.2 Definition Format

Bold directives are required, while the others are optional.

define servicegroup{	
servicegroup_name	***servicegroup_name***
alias	***alias***
members	*services*
servicegroup_members	*servicegroups*
notes	*note_string*
notes_url	*url*
action_url	*url*
}	

8.4.3 Example Definition

```
define servicegroup{
servicegroup_name       dbservices
alias                   Database Services
members                 ms1,SQL Server,ms1,SQL Serverc Agent,ms1,SQL DTC
}
```

8.4.4 Directive Descriptions:

- servicegroup_name

This directive is used to define a short name used to identify the service group.

- alias

This directive is used to define is a longer name or description used to identify the service group. It is provided in order to allow you to more easily identify a particular service group.

- members

This is a list of the *descriptions* of *services* (and the names of their corresponding hosts) that should be included in this group. Host and service names should be separated by commas. This directive may be used as an alternative to the *servicegroups* directive in *service definitions*. The format of the member directive is as follows (note that a host name must precede a service name/description):

```
members=<host1>,<service1>,<host2>,<service2>,...,<host*n*>,<service*n*>
```

- servicegroup_members

This optional directive can be used to include services from other "sub" service groups in this service group. Specify a comma-delimited list of short names of other service groups whose members should be included in this group.

- notes

This directive is used to define an optional string of notes pertaining to the service group. If you specify a note here, you will see the it in the extended information CGI (when you are viewing information about the specified service group).

- notes_url

This directive is used to define an optional URL that can be used to provide more information about the service group. If you specify an URL, you will see a red folder icon in the CGIs (when you are viewing service group information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will the the same as what is used to access the CGIs (i.e. ///cgi-bin/shinken///). This can be very useful if you want to make detailed information on the service group, emergency contact methods, etc. available to other support staff.

- action_url

This directive is used to define an optional URL that can be used to provide more actions to be performed on the service group. If you specify an URL, you will see a red "splat" icon in the CGIs (when you are viewing service group information) that links to the URL you specify here. Any valid URL can be used. If you plan on using relative paths, the base path will the the same as what is used to access the CGIs (i.e. ///cgi-bin/shinken///).

8.5 Contact Definition

8.5.1 Description

A contact definition is used to identify someone who should be contacted in the event of a problem on your network. The different arguments to a contact definition are described below.

8.5.2 Definition Format

Bold directives are required, while the others are optional.

define contact{	
contact_name	***contact_name***
alias	*alias*
contactgroups	*contactgroup_names*
host_notifications_enabled	**[0/1]**
service_notifications_enabled	**[0/1]**
host_notification_period	***timeperiod_name***
service_notification_period	***timeperiod_name***
host_notification_options	**[d,u,r,f,s,n]**
service_notification_options	**[w,u,c,r,f,s,n]**
host_notification_commands	***command_name***
service_notification_commands	***command_name***
email	*email_address*
pager	*pager_number or pager_email_gateway*
address*x*	*additional_contact_address*
can_submit_commands	[0/1]
is_admin	[0/1]
retain_status_information	[0/1]
retain_nonstatus_information	[0/1]
min_business_impact	[0/1/2/3/4/5]
}	

8.5.3 Example Definition

```
define contact{
contact_name                    jdoe
alias                           John Doe
host_notifications_enabled      1
service_notifications_enabled   1
service_notification_period     24x7
host_notification_period        24x7
service_notification_options    w,u,c,r
host_notification_options       d,u,r
service_notification_commands   notify-by-email
host_notification_commands      host-notify-by-email
```

```
email                           jdoe@localhost.localdomain
pager                           555-5555@pagergateway.localhost.localdomain
address1                        xxxxx.xyyy@icq.com
address2                        555-555-5555
can_submit_commands             1
}
```

8.5.4 Directive Descriptions

- contact_name

This directive is used to define a short name used to identify the contact. It is referenced in *contact group* definitions. Under the right circumstances, the $CONTACTNAME$ *macro* will contain this value.

- alias

This directive is used to define a longer name or description for the contact. Under the rights circumstances, the $CONTACTALIAS$ *macro* will contain this value. If not specified, the *contact_name* will be used as the alias.

- contactgroups

This directive is used to identify the *short name(s)* of the *contactgroup(s)* that the contact belongs to. Multiple contactgroups should be separated by commas. This directive may be used as an alternative to (or in addition to) using the *members* directive in *contactgroup* definitions.

- host_notifications_enabled

This directive is used to determine whether or not the contact will receive notifications about host problems and recoveries. Values :

- 0 = don't send notifications
- 1 = send notifications
- service_notifications_enabled

This directive is used to determine whether or not the contact will receive notifications about service problems and recoveries. Values:

- 0 = don't send notifications
- 1 = send notifications
- host_notification_period

This directive is used to specify the short name of the *time period* during which the contact can be notified about host problems or recoveries. You can think of this as an "on call" time for host notifications for the contact. Read the documentation on *time periods* for more information on how this works and potential problems that may result from improper use.

- service_notification_period

This directive is used to specify the short name of the *time period* during which the contact can be notified about service problems or recoveries. You can think of this as an "on call" time for service notifications for the contact. Read the documentation on *time periods* for more information on how this works and potential problems that may result from improper use.

- host_notification_commands

This directive is used to define a list of the *short names* of the *commands* used to notify the contact of a *host* problem or recovery. Multiple notification commands should be separated by commas. All notification commands are executed

when the contact needs to be notified. The maximum amount of time that a notification command can run is controlled by the notification_timeout option.

- host_notification_options

This directive is used to define the host states for which notifications can be sent out to this contact. Valid options are a combination of one or more of the following:

- d = notify on DOWN host states
- u = notify on UNREACHABLE host states
- r = notify on host recoveries (UP states)
- f = notify when the host starts and stops *flapping*,
- s = send notifications when host or service *scheduled downtime* starts and ends. If you specify **n** (none) as an option, the contact will not receive any type of host notifications.
- service_notification_options

This directive is used to define the service states for which notifications can be sent out to this contact. Valid options are a combination of one or more of the following:

- w = notify on WARNING service states
- u = notify on UNKNOWN service states
- c = notify on CRITICAL service states
- r = notify on service recoveries (OK states)
- f = notify when the service starts and stops *flapping*.
- n = (none) : the contact will not receive any type of service notifications.
- service_notification_commands

This directive is used to define a list of the *short names* of the commands used to notify the contact of a *service* problem or recovery. Multiple notification commands should be separated by commas. All notification commands are executed when the contact needs to be notified. The maximum amount of time that a notification command can run is controlled by the notification_timeout option.

- email

This directive is used to define an email address for the contact. Depending on how you configure your notification commands, it can be used to sendout an alert email to the contact. Under the right circumstances, the $CONTACTEMAIL$ *macro* will contain this value.

- pager

This directive is used to define a pager number for the contact. It can also be an email address to a pager gateway (i.e. pagejoe@pagenet.com). Depending on how you configure your notification commands, it can be used to send out an alert page to the contact. Under the right circumstances, the $CONTACTPAGER$ *macro* will contain this value.

- address*x*

Address directives are used to define additional "addresses" for the contact. These addresses can be anything - cell phone numbers, instant messaging addresses, etc. Depending on how you configure your notification commands, they can be used to send out an alert o the contact. Up to six addresses can be defined using these directives (*address1* through *address6*). The $CONTACTADDRESS*x*$ *macro* will contain this value.

- can_submit_commands

This directive is used to determine whether or not the contact can submit *external commands* to Shinken from the CGIs. Values:

- 0 = don't allow contact to submit commands
- 1 = allow contact to submit commands.
- is_admin

This directive is used to determine whether or not the contact can see all object in *WebUI*. Values:

- 0 = normal user, can see all objects he is in contact
- 1 = allow contact to see all objects
- retain_status_information

This directive is used to determine whether or not status-related information about the contact is retained across program restarts. This is only useful if you have enabled state retention using the *retain_state_information* directive. Value :

- 0 = disable status information retention
- 1 = enable status information retention.
- retain_nonstatus_information

This directive is used to determine whether or not non-status information about the contact is retained across program restarts. This is only useful if you have enabled state retention using the *retain_state_information* directive. Value :

- 0 = disable non-status information retention
- 1 = enable non-status information retention
- min_business_impact

This directive is use to define the minimum business criticity level of a service/host the contact will be notified. Please see *root_problems_and_impacts* for more details.

- 0 = less important
- 1 = more important than 0
- 2 = more important than 1
- 3 = more important than 2
- 4 = more important than 3
- 5 = most important

8.6 Contact Group Definition

8.6.1 Description

A contact group definition is used to group one or more *contacts* together for the purpose of sending out alert/recovery *notifications*.

8.6.2 Definition Format:

Bold directives are required, while the others are optional.

define contactgroup{	
contactgroup_name	**contactgroup_name**
alias	**alias**
members	*contacts*
contactgroup_members	*contactgroups*
}	

8.6.3 Example Definition:

```
define contactgroup{
contactgroup_name       novell-admins
alias                   Novell Administrators
members                 jdoe,rtobert,tzach
}
```

8.6.4 Directive Descriptions:

contactgroup_name

This directive is a short name used to identify the contact group.

alias

This directive is used to define a longer name or description used to identify the contact group.

members

This directive is used to define a list of the *short names* of contacts that should be included in this group. Multiple contact names should be separated by commas. This directive may be used as an alternative to (or in addition to) using the *contactgroups* directive in contact definitions.

contactgroup_members

This optional directive can be used to include contacts from other "sub" contact groups in this contact group. Specify a comma-delimited list of short names of other contact groups whose members should be included in this group.

8.7 Time Period Definition

8.7.1 Description

A time period is a list of times during various days that are considered to be "valid" times for notifications and service checks. It consists of time ranges for each day of the week that "rotate" once the week has come to an end. Different types of exceptions to the normal weekly time are supported, including: specific weekdays, days of generic months, days of specific months, and calendar dates.

8.7.2 Definition Format

Bold directives are required, while the others are optional.

define timeperiod{	
timeperiod_name	***timeperiod_name***
alias	***alias***
[weekday]	*timeranges*
[exception]	*timeranges*
exclude	[*timeperiod1,timeperiod2,...,timeperiodn*]
}	

8.7.3 Example Definitions

```
define timeperiod{
timeperiod_name         nonworkhours
alias                   Non-Work Hours
sunday                  00:00-24:00              ; Every Sunday of every week
monday                  00:00-09:00,17:00-24:00  ; Every Monday of every week
tuesday                 00:00-09:00,17:00-24:00  ; Every Tuesday of every week
wednesday               00:00-09:00,17:00-24:00  ; Every Wednesday of every week
thursday                00:00-09:00,17:00-24:00  ; Every Thursday of every week
friday                  00:00-09:00,17:00-24:00  ; Every Friday of every week
saturday                00:00-24:00              ; Every Saturday of every week
}

define timeperiod{
timeperiod_name         misc-single-days
alias                   Misc Single Days
1999-01-28              00:00-24:00              ; January 28th, 1999
monday 3                00:00-24:00              ; 3rd Monday of every month
day 2                   00:00-24:00              ; 2nd day of every month
february 10             00:00-24:00              ; February 10th of every year
february -1             00:00-24:00              ; Last day in February of every year
friday -2               00:00-24:00              ; 2nd to last Friday of every month
thursday -1 november    00:00-24:00              ; Last Thursday in November of every year
}

define timeperiod{
timeperiod_name                 misc-date-ranges
alias                           Misc Date Ranges
2007-01-01 - 2008-02-01         00:00-24:00      ; January 1st, 2007 to February 1st, 2008
monday 3 - thursday 4           00:00-24:00      ; 3rd Monday to 4th Thursday of every month
day 1 - 15                      00:00-24:00      ; 1st to 15th day of every month
day 20 - -1                     00:00-24:00      ; 20th to the last day of every month
july 10 - 15                    00:00-24:00      ; July 10th to July 15th of every year
april 10 - may 15               00:00-24:00      ; April 10th to May 15th of every year
tuesday 1 april - friday 2 may  00:00-24:00      ; 1st Tuesday in April to 2nd Friday in May of every
}

define timeperiod{
timeperiod_name                      misc-skip-ranges
alias                                Misc Skip Ranges
2007-01-01 - 2008-02-01 / 3          00:00-24:00    ; Every 3 days from January 1st, 2007 to February
2008-04-01 / 7                       00:00-24:00    ; Every 7 days from April 1st, 2008 (continuing i
monday 3 - thursday 4 / 2            00:00-24:00    ; Every other day from 3rd Monday to 4th Thursday
day 1 - 15 / 5                       00:00-24:00    ; Every 5 days from the 1st to the 15th day of ev
july 10 - 15 / 2                     00:00-24:00    ; Every other day from July 10th to July 15th of
tuesday 1 april - friday 2 may / 6   00:00-24:00    ; Every 6 days from the 1st Tuesday in April to t
}
```

8.7.4 Directive Descriptions

timeperiod_name

This directives is the short name used to identify the time period.

alias

This directive is a longer name or description used to identify the time period.

[weekday]

The weekday directives ("*sunday*" through "*saturday*")are comma-delimited lists of time ranges that are "valid" times for a particular day of the week. Notice that there are seven different days for which you can define time ranges (Sunday through Saturday). Each time range is in the form of **HH:MM-HH:MM**, where hours are Specified on a 24 hour clock. For example, **00:15-24:00** means 12:15am in the morning for this day until 12:00am midnight (a 23 hour, 45 minute total time range). If you wish to exclude an entire day from the timeperiod, simply do not include it in the timeperiod definition.

The daterange format are multiples :

- Calendar Daterange : look like a standard date, so like 2005-04-04 - 2008-09-19.
- Month Week Day: Then there are the month week day daterange same than before, but without the year and with day names That give something like : tuesday 2 january - thursday 4 august / 5
- Now Month Date Daterange: It looks like : february 1 - march 15 / 3
- Now Month Day Daterange. It looks like day 13 - 14
- Now Standard Daterange: Ok this time it's quite easy: monday

  [exception]

You can specify several different types of exceptions to the standard rotating weekday schedule. Exceptions can take a number of different forms including single days of a specific or generic month, single weekdays in a month, or single calendar dates. You can also specify a range of days/dates and even specify skip intervals to obtain functionality described by "every 3 days between these dates". Rather than list all the possible formats for exception strings, I'll let you look at the example timeperiod definitions above to see what's possible. :-) Weekdays and different types of exceptions all have different levels of precedence, so its important to understand how they can affect each other. More information on this can be found in the documentation on *timeperiods*.

exclude

This directive is used to specify the short names of other timeperiod definitions whose time ranges should be excluded from this timeperiod. Multiple timeperiod names should be separated with a comma.

Note: The day skip functionality is not managed from now, so it's like all is / 1

8.8 Command Definition

8.8.1 Description

A command definition is just that. It defines a command. Commands that can be defined include service checks, service notifications, service event handlers, host checks, host notifications, and host event handlers. Command definitions can contain *macros*, but you must make sure that you include only those macros that are "valid" for the circumstances when the command will be used. More information on what macros are available and when they are "valid" can be found *here*. The different arguments to a command definition are outlined below.

Tip: If, you need to have the '$' character in one of your command (and not referring to a macro), please put "$$" instead. Shinken will replace it well

8.8.2 Definition Format

Bold directives are required, while the others are optional.

define command{	
command_name	***command_name***
command_line	***command_line***
poller_tag	*poller_tag*
}	

8.8.3 Example Definition

```
define command{
  command_name   check_pop
  command_line   /usr/local/shinken/libexec/check_pop -H $HOSTADDRESS$
}
```

8.8.4 Directive Descriptions

command_name

This directive is the short name used to identify the command. It is referenced in *contact*, *host*, and *service* definitions (in notification, check, and event handler directives), among other places.

command_line

This directive is used to define what is actually executed by Shinken when the command is used for service or host checks, notifications, or *event handlers*. Before the command line is executed, all valid *macros* are replaced with their respective values. See the documentation on macros for determining when you can use different macros. Note that the command line is *not* surrounded in quotes. Also, if you want to pass a dollar sign ($) on the command line, you have to escape it with another dollar sign.

You may not include a **semicolon** (;) in the *command_line* directive, because everything after it will be ignored as a config file comment. You can work around this limitation by setting one of the $USER$ macros in your *resource file* to a semicolon and then referencing the appropriate $USER$ macro in the *command_line* directive in place of the semicolon.

If you want to pass arguments to commands during runtime, you can use $ARGn$ macros in the *command_line* directive of the command definition and then separate individual arguments from the command name (and from each other) using bang (!) characters in the object definition directive (host check command, service event handler command, etc.) that references the command. More information on how arguments in command definitions are processed during runtime can be found in the documentation on *macros*.

poller_tag

This directive is used to define the poller_tag of this command. If the host/service that call this command do nto override it with their own poller_tag, it will make this command if used in a check only taken by polelrs that also have this value in their poller_tags parameter.

By default there is no poller_tag, so all untagged pollers can take it.

8.9 Service Dependency Definition

8.9.1 Description

Service dependencies are an advanced feature of Shinken that allow you to suppress notifications and active checks of services based on the status of one or more other services. Service dependencies are optional and are mainly targeted at advanced users who have complicated monitoring setups. More information on how service dependencies work (read this!) can be found *here*.

8.9.2 Definition Format

Bold directives are required, while the others are optional. However, you must supply at least one type of criteria for the definition to be of much use.

define servicedependency{	
dependent_host_name	***host_name***
dependent_hostgroup_name	*hostgroup_name*
dependent_service_description	***service_description***
host_name	***host_name***
hostgroup_name	*hostgroup_name*
service_description	***service_description***
inherits_parent	[0/1]
execution_failure_criteria	[o,w,u,c,p,n]
notification_failure_criteria	[o,w,u,c,p,n]
dependency_period	timeperiod_name
}	

8.9.3 Example Definition

```
define servicedependency{
host_name                       WWW1
service_description             Apache Web Server
dependent_host_name             WWW1
dependent_service_description   Main Web Site
execution_failure_criteria      n
notification_failure_criteria   w,u,c
}
```

8.9.4 Directive Descriptions

dependent_host_name

This directive is used to identify the *short name(s)* of the *host(s)* that the *dependent* service "runs" on or is associated with. Multiple hosts should be separated by commas. Leaving this directive blank can be used to create "same host" dependencies.

dependent_hostgroup

This directive is used to specify the *short name(s)* of the *hostgroup(s)* that the *dependent* service "runs" on or is associated with. Multiple hostgroups should be separated by commas. The "dependent_hostgroup" may be used instead of, or in addition to, the "dependent_host" directive.

dependent_service_description

This directive is used to identify the *description* of the *dependent service*.

host_name

This directive is used to identify the *short name(s)* of the *host(s)* that the service *that is being depended upon* (also referred to as the master service) "runs" on or is associated with. Multiple hosts should be separated by commas.

hostgroup_name

This directive is used to identify the *short name(s)* of the *hostgroup(s)* that the service *that is being depended upon* (also referred to as the master service) "runs" on or is associated with. Multiple hostgroups should be separated by commas. The "hostgroup_name" may be used instead of, or in addition to, the "host_name" directive.

service_description

This directive is used to identify the *description* of the *service that is being depended upon* (also referred to as the master service).

inherits_parent

This directive indicates whether or not the dependency inherits dependencies of the service *that is being depended upon* (also referred to as the master service). In other words, if the master service is dependent upon other services and any one of those dependencies fail, this dependency will also fail.

execution_failure_criteria

This directive is used to specify the criteria that determine when the dependent service should *not* be actively checked. If the *master* service is in one of the failure states we specify, the *dependent* service will not be actively checked. Valid options are a combination of one or more of the following (multiple options are separated with commas):

- **o** = fail on an OK state
- **w** = fail on a WARNING state
- **u** = fail on an UNKNOWN state
- **c** = fail on a CRITICAL state
- **p** = fail on a pending state (e.g. the service has not yet been checked).
- **n** (none) : the execution dependency will never fail and checks of the dependent service will always be actively checked (if other conditions allow for it to be).

If you specify **o,c,u** in this field, the *dependent* service will not be actively checked if the *master* service is in either an OK, a CRITICAL, or an UNKNOWN state.

notification_failure_criteria

This directive is used to define the criteria that determine when notifications for the dependent service should *not* be sent out. If the *master* service is in one of the failure states we specify, notifications for the *dependent* service will not be sent to contacts. Valid options are a combination of one or more of the following:

- **o** = fail on an OK state
- **w** = fail on a WARNING state
- **u** = fail on an UNKNOWN state
- **c** = fail on a CRITICAL state
- **p** = fail on a pending state (e.g. the service has not yet been checked).
- **n** = (none) : the notification dependency will never fail and notifications for the dependent service will always be sent out.

If you specify **w** in this field, the notifications for the *dependent* service will not be sent out if the *master* service is in a WARNING state.

dependency_period

This directive is used to specify the short name of the *time period* during which this dependency is valid. If this directive is not specified, the dependency is considered to be valid during all times.

8.10 Service Escalation Definition

8.10.1 Description

Service escalations are completely optional and are used to escalate notifications for a particular service. More information on how notification escalations work can be found *here*.

8.10.2 Definition Format

Bold directives are required, while the others are optional.

define serviceescalation{	
host_name	***host_name***
hostgroup_name	*hostgroup_name*
service_description	***service_description***
contacts	***contacts***
contact_groups	***contactgroup_name***
first_notification	**#**
last_notification	**#**
first_notification_time	#
last_notification_time	#
notification_interval	**#**
escalation_period	timeperiod_name
escalation_options	[w,u,c,r]
}	

8.10.3 Example Definition

Here for an scalation that will escalade to "themanagers" after one hour problem and ends at 2 hours.

```
define serviceescalation{
host_name                nt-3
service_description      Processor Load
first_notification_time      60
last_notification_time       120
notification_interval    30
contact_groups           themanagers
}
```

8.10.4 Directive Descriptions

host_name

This directive is used to identify the *short name(s)* of the *host(s)* that the *service* escalation should apply to or is associated with.

hostgroup_name

This directive is used to specify the *short name(s)* of the *hostgroup(s)* that the service escalation should apply to or is associated with. Multiple hostgroups should be separated by commas. The "hostgroup_name" may be used instead of, or in addition to, the "host_name" directive.

service_description

This directive is used to identify the *description* of the *service* the escalation should apply to.

first_notification

This directive is a number that identifies the *first* notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if the service is in a non-OK state long enough for a third notification to go out.

last_notification

This directive is a number that identifies the *last* notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five notifications are sent out for the service. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).

first_notification_time

This directive is the number of "time intervals" (60 seconds by default) until that makes the *first* notification for which this escalation is effective. For instance, if you set this value to 60, this escalation will only be used if the service is in a non-OK state long enough for 60 minutes notification to go out.

last_notification_time

This directive is a number of "time intervals" (60 seconds by default) until that makes the *last* notification for which this escalation is effective. For instance, if you set this value to 120, this escalation will not be used if more than two hours after then notifications are sent out for the service. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).

contacts

This is a list of the *short names* of the *contacts* that should be notified whenever there are problems (or recoveries) with this service. Multiple contacts should be separated by commas. Useful if you want notifications to go to just a few people and don't want to configure *contact groups*. You must specify at least one contact or contact group in each service escalation definition.

contact_groups

This directive is used to identify the *short name* of the *contact group* that should be notified when the service notification is escalated. Multiple contact groups should be separated by commas. You must specify at least one contact or contact group in each service escalation definition.

notification_interval

This directive is used to determine the interval at which notifications should be made while this escalation is valid. If you specify a value of 0 for the interval, Shinken will send the first notification when this escalation definition is valid, but will then prevent any more problem notifications from being sent out for the host. Notifications are sent out again until the host recovers. This is useful if you want to stop having notifications sent out after a certain amount of time.

If multiple escalation entries for a host overlap for one or more notification ranges, the smallest notification interval from all escalation entries is used.

escalation_period

This directive is used to specify the short name of the *time period* during which this escalation is valid. If this directive is not specified, the escalation is considered to be valid during all times.

escalation_options

This directive is used to define the criteria that determine when this service escalation is used. The escalation is used only if the service is in one of the states specified in this directive. If this directive is not specified in a service escalation, the escalation is considered to be valid during all service states. Valid options are a combination of one or more of the following:

- **r** = escalate on an OK (recovery) state
- **w** = escalate on a WARNING state
- **u** = escalate on an UNKNOWN state
- **c** = escalate on a CRITICAL state

If you specify **w** in this field, the escalation will only be used if the service is in a WARNING state.

8.11 Host Dependency Definition

8.11.1 Description

Host dependencies are an advanced feature of Shinken that allow you to suppress notifications for hosts based on the status of one or more other hosts. Host dependencies are optional and are mainly targeted at advanced users who have complicated monitoring setups. More information on how host dependencies work (read this!) can be found *here*.

8.11.2 Definition Format

Bold directives are required, while the others are optional.

define hostdependency{	
dependent_host_name	***host_name***
dependent_hostgroup_name	*hostgroup_name*
host_name	***host_name***
hostgroup_name	*hostgroup_name*
inherits_parent	[0/1]
execution_failure_criteria	[o,d,u,p,n]
notification_failure_criteria	[o,d,u,p,n]
dependency_period	timeperiod_name
}	

8.11.3 Example Definition

```
define hostdependency{
host_name                       WWW1
dependent_host_name             DBASE1
notification_failure_criteria   d,u
}
```

8.11.4 Directive Descriptions

dependent_host_name

This directive is used to identify the *short name(s)* of the *dependent host(s)*. Multiple hosts should be separated by commas.

dependent_hostgroup_name

This directive is used to identify the *short name(s)* of the *dependent hostgroup(s)*. Multiple hostgroups should be separated by commas. The dependent_hostgroup_name may be used instead of, or in addition to, the dependent_host_name directive.

host_name

This directive is used to identify the *short name(s)* of the *host(s) that is being depended upon* (also referred to as the master host). Multiple hosts should be separated by commas.

hostgroup_name

This directive is used to identify the *short name(s)* of the *hostgroup(s) that is being depended upon* (also referred to as the master host). Multiple hostgroups should be separated by commas. The hostgroup_name may be used instead of, or in addition to, the host_name directive.

inherits_parent

This directive indicates whether or not the dependency inherits dependencies of the host *that is being depended upon* (also referred to as the master host). In other words, if the master host is dependent upon other hosts and any one of those dependencies fail, this dependency will also fail.

execution_failure_criteria

This directive is used to specify the criteria that determine when the dependent host should *not* be actively checked. If the *master* host is in one of the failure states we specify, the *dependent* host will not be actively checked. Valid options are a combination of one or more of the following (multiple options are separated with commas):

- **o** = fail on an UP state
- **d** = fail on a DOWN state
- **u** = fail on an UNREACHABLE state
- **p** = fail on a pending state (e.g. the host has not yet been checked)
- **n** (none) : the execution dependency will never fail and the dependent host will always be actively checked (if other conditions allow for it to be).

If you specify **u,d** in this field, the *dependent* host will not be actively checked if the *master* host is in either an UNREACHABLE or DOWN state.

notification_failure_criteria

This directive is used to define the criteria that determine when notifications for the dependent host should *not* be sent out. If the *master* host is in one of the failure states we specify, notifications for the *dependent* host will not be sent to contacts. Valid options are a combination of one or more of the following:

- **o** = fail on an UP state
- **d** = fail on a DOWN state
- **u** = fail on an UNREACHABLE state
- **p** = fail on a pending state (e.g. the host has not yet been checked)
- **n** = (none) : the notification dependency will never fail and notifications for the dependent host will always be sent out.

If you specify **d** in this field, the notifications for the *dependent* host will not be sent out if the *master* host is in a DOWN state.

dependency_period

This directive is used to specify the short name of the *time period* during which this dependency is valid. If this directive is not specified, the dependency is considered to be valid during all times.

8.12 Host Escalation Definition

8.12.1 Description

Host escalations are completely optional and are used to escalate notifications for a particular host. More information on how notification escalations work can be found *here*.

8.12.2 Definition Format

Bold directives are required, while the others are optional.

define hostescalation{	
host_name	***host_name***
hostgroup_name	*hostgroup_name*
contacts	***contacts***
contact_groups	***contactgroup_name***
first_notification	**#**
last_notification	**#**
first_notification_time	#
last_notification_time	#
notification_interval	**#**
escalation_period	timeperiod_name
escalation_options	[d,u,r]
}	

8.12.3 Example Definition

Escalate to all-router-admins after one hour of problem, and stop at 2 hours.

```
define hostescalation{
  host_name                router-34
  first_notification       60
  last_notification        120
  notification_interval    60
  contact_groups           all-router-admins
}
```

8.12.4 Directive Descriptions

host_name

This directive is used to identify the *short name* of the *host* that the escalation should apply to.

hostgroup_name

This directive is used to identify the *short name(s)* of the *hostgroup(s)* that the escalation should apply to. Multiple hostgroups should be separated by commas. If this is used, the escalation will apply to all hosts that are members of the specified hostgroup(s).

first_notification

This directive is a number that identifies the *first* notification for which this escalation is effective. For instance, if you set this value to 3, this escalation will only be used if the host is down or unreachable long enough for a third notification to go out.

last_notification

This directive is a number that identifies the *last* notification for which this escalation is effective. For instance, if you set this value to 5, this escalation will not be used if more than five notifications are sent out for the host. Setting this value to 0 means to keep using this escalation entry forever (no matter how many notifications go out).

first_notification_time

This directive is the number of "time intervals" (60 seconds by default) until that makes the *first* notification for which this escalation is effective. For instance, if you set this value to 60, this escalation will only be used if the host is in a non-OK state long enough for 60 minutes notification to go out.

last_notification_time

This directive is a number of "time intervals" (60 seconds by default) until that makes the *last* notification for which this escalati

contacts

This is a list of the *short names* of the *contacts* that should be notified whenever there are problems (or recoveries) with this host. Multiple contacts should be separated by commas. Useful if you want notifications to go to just a few people and don't want to configure *contact groups*. You must specify at least one contact or contact group in each host escalation definition.

contact_groups

This directive is used to identify the *short name* of the *contact group* that should be notified when the host notification is escalated. Multiple contact groups should be separated by commas. You must specify at least one contact or contact group in each host escalation definition.

notification_interval

This directive is used to determine the interval at which notifications should be made while this escalation is valid. If you specify a value of 0 for the interval, Shinken will send the first notification when this escalation definition is valid, but will then prevent any more problem notifications from being sent out for the host. Notifications are sent out again until the host recovers. This is useful if you want to stop having notifications sent out after a certain amount of time.

If multiple escalation entries for a host overlap for one or more notification ranges, the smallest notification interval from all escalation entries is used.

escalation_period

This directive is used to specify the short name of the *time period* during which this escalation is valid. If this directive is not specified, the escalation is considered to be valid during all times.

escalation_options

This directive is used to define the criteria that determine when this host escalation is used. The escalation is used only if the host is in one of the states specified in this directive. If this directive is not specified in a host escalation, the escalation is considered to be valid during all host states. Valid options are a combination of one or more of the following :

- **r** = escalate on an UP (recovery) state
- **d** = escalate on a DOWN state
- **u** = escalate on an UNREACHABLE state

If you specify **d** in this field, the escalation will only be used if the host is in a DOWN state.

8.13 Extended Host Information Definition

8.13.1 Description

Extended host information entries are basically used to make the output from the status, statusmap, statuswrl, and extinfo CGIs look pretty. They have no effect on monitoring and are completely optional.

Note: Tip: As of Nagios 3.x, all directives contained in extended host information definitions are also available in host definitions. Thus, you can choose to define the directives below in your host definitions if it makes your configuration simpler. Separate extended host information definitions will continue to be supported for backward compatability.

8.13.2 Definition Format

Bold directives are required, while the others are optional.

define hostextinfo{	
host_name	***host_name***
notes	*notes*
notes_url	*notes_url*
action_url	*action_url*
icon_image	*image_file*
icon_image_alt	*alt_string*
vrml_image	*image_file*
statusmap_image	*image_file*
2d_coords	*x_coord,y_coord*
3d_coords	*x_coord,y_coord,z_coord*
}	

8.13.3 Example Definition

define hostextinfo{ host_name netware1 notes This is the primary Netware file server notes_url http://webserver.localhost.localdomain/hostinfo.pl?host=netware1 icon_image novell40.png icon_image_alt IntranetWare 4.11 vrml_image novell40.png statusmap_image novell40.gd2 2d_coords 100,250 3d_coords 100.0,50.0,75.0 }

8.13.4 Directive Descriptions

host_name

This variable is used to identify the short name of the host which the data is associated with.

notes

This directive is used to define an optional string of notes pertaining to the host. If you specify a note here, you will see the it in the extended information CGI (when you are viewing information about the specified host).

notes_url

This variable is used to define an optional URL that can be used to provide more information about the host. If you specify an URL, you will see a link that says "Extra Host Notes" in the extended information CGI (when you are viewing information about the specified host). Any valid URL can be used. If you plan on using relative paths, the

base path will the the same as what is used to access the CGIs (i.e. /cgi-bin/nagios/). This can be very useful if you want to make detailed information on the host, emergency contact methods, etc. available to other support staff.

action_url

This directive is used to define an optional URL that can be used to provide more actions to be performed on the host. If you specify an URL, you will see a link that says "Extra Host Actions" in the extended information CGI (when you are viewing information about the specified host). Any valid URL can be used. If you plan on using relative paths, the base path will the the same as what is used to access the CGIs (i.e. /cgi-bin/nagios/).

icon_image

This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be displayed in the status and extended information CGIs. The image will look best if it is 40x40 pixels in size. Images for hosts are assumed to be in the logos/ subdirectory in your HTML images directory (i.e. /usr/local/nagios/share/images/logos).

icon_image_alt

This variable is used to define an optional string that is used in the ALT tag of the image specified by the <*icon_image*> argument. The ALT tag is used in the status, extended information and statusmap CGIs.

vrml_image

This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be used as the texture map for the specified host in the statuswrl CGI. Unlike the image you use for the <icon_image> variable, this one should probably not have any transparency. If it does, the host object will look a bit wierd. Images for hosts are assumed to be in the logos/ subdirectory in your HTML images directory (i.e. /usr/local/nagios/share/images/logos).

statusmap_image

This variable is used to define the name of an image that should be associated with this host in the statusmap CGI. You can specify a JPEG, PNG, and GIF image if you want, although I would strongly suggest using a GD2 format image, as other image formats will result in a lot of wasted CPU time when the statusmap image is generated. GD2 images can be created from PNG images by using the pngtogd2 utility supplied with Thomas Boutell's gd library. The GD2 images should be created in uncompressed format in order to minimize CPU load when the statusmap CGI is generating the network map image. The image will look best if it is 40x40 pixels in size. You can leave these option blank if you are not using the statusmap CGI. Images for hosts are assumed to be in the logos/ subdirectory in your HTML images directory (i.e. /usr/local/nagios/share/images/logos).

2d_coords

This variable is used to define coordinates to use when drawing the host in the statusmap CGI. Coordinates should be given in positive integers, as they correspond to physical pixels in the generated image. The origin for drawing (0,0) is in the upper left hand corner of the image and extends in the positive x direction (to the right) along the top of the image and in the positive y direction (down) along the left hand side of the image. For reference, the size of the icons drawn is usually about 40x40 pixels (text takes a little extra space). The coordinates you specify here are for the upper left hand corner of the host icon that is drawn. Note: Don't worry about what the maximum x and y coordinates that you can use are. The CGI will automatically calculate the maximum dimensions of the image it creates based on the largest x and y coordinates you specify.

3d_coords

This variable is used to define coordinates to use when drawing the host in the statuswrl CGI. Coordinates can be positive or negative real numbers. The origin for drawing is (0.0,0.0,0.0). For reference, the size of the host cubes drawn is 0.5 units on each side (text takes a little more space). The coordinates you specify here are used as the center of the host cube.

8.14 Extended Service Information Definition

8.14.1 Description

Extended service information entries are basically used to make the output from the status and extinfo CGIs look pretty. They have no effect on monitoring and are completely optional.

Note: Tip: As of Nagios 3.x, all directives contained in extended service information definitions are also available in service definitions. Thus, you can choose to define the directives below in your service definitions if it makes your configuration simpler. Separate extended service information definitions will continue to be supported for backward compatability.

8.14.2 Definition Format

Bold directives are required, while the others are optional.

define serviceextinfo{	
host_name	***host_name***
service_description	***service_description***
notes	*notes*
notes_url	*notes_url*
action_url	*action_url*
icon_image	*image_file*
icon_image_alt	*alt_string*
}	

8.14.3 Example Definition

define serviceextinfo{ host_name linux2 service_description Log Anomalies notes Security-related log anomalies on secondary Linux server notes_url http://webserver.localhost.localdomain/serviceinfo.pl?host=linux2&service=Log+Anomalies icon_image security.png icon_image_alt Security-Related Alerts }

8.14.4 Directive Descriptions

host_name

This directive is used to identify the short name of the host that the service is associated with.

service_description

This directive is description of the service which the data is associated with.

notes

This directive is used to define an optional string of notes pertaining to the service. If you specify a note here, you will see the it in the extended information CGI (when you are viewing information about the specified service).

notes_url

This directive is used to define an optional URL that can be used to provide more information about the service. If you specify an URL, you will see a link that says "Extra Service Notes" in the extended information CGI (when you are viewing information about the specified service). Any valid URL can be used. If you plan on using relative paths, the

base path will the the same as what is used to access the CGIs (i.e. /cgi-bin/nagios/). This can be very useful if you want to make detailed information on the service, emergency contact methods, etc. available to other support staff.

action_url

This directive is used to define an optional URL that can be used to provide more actions to be performed on the service. If you specify an URL, you will see a link that says "Extra Service Actions" in the extended information CGI (when you are viewing information about the specified service). Any valid URL can be used. If you plan on using relative paths, the base path will the the same as what is used to access the CGIs (i.e. /cgi-bin/nagios/).

icon_image

This variable is used to define the name of a GIF, PNG, or JPG image that should be associated with this host. This image will be displayed in the status and extended information CGIs. The image will look best if it is 40x40 pixels in size. Images for hosts are assumed to be in the logos/ subdirectory in your HTML images directory (i.e. /usr/local/nagios/share/images/logos).

icon_image_alt

This variable is used to define an optional string that is used in the ALT tag of the image specified by the *<icon_image>* argument. The ALT tag is used in the status, extended information and statusmap CGIs.

8.15 Notification Way Definition

8.15.1 Description

A notificationway definition is used to define a way a contact if notified.

8.15.2 Definition Format

Bold directives are required, while the others are optional.

define notificationway{	
notificationway_name	***notificationway_name***
host_notification_period	***timeperiod_name***
service_notification_period	***timeperiod_name***
host_notification_options	**[d,u,r,f,s,n]**
service_notification_options	**[w,u,c,r,f,s,n]**
host_notification_commands	***command_name***
service_notification_commands	***command_name***
min_business_impact	[0/1/2/3/4/5]
}	

8.15.3 Example Definition

```
# Email the whole 24x7 is okay
define notificationway{
notificationway_name            email_in_day
service_notification_period     24x7
host_notification_period        24x7
service_notification_options    w,u,c,r,f
host_notification_options       d,u,r,f,s
service_notification_commands   notify-service
```

```
host_notification_commands      notify-host
}
```

8.15.4 Directive Descriptions

notificationway_name

This directive define the name of the notification witch be specified further in a contact definition

host_notification_period

This directive is used to specify the short name of the *time period* during which the contact can be notified about host problems or recoveries. You can think of this as an "on call" time for host notifications for the contact. Read the documentation on *time periods* for more information on how this works and potential problems that may result from improper use.

service_notification_period

This directive is used to specify the short name of the *time period* during which the contact can be notified about service problems or recoveries. You can think of this as an "on call" time for service notifications for the contact. Read the documentation on *time periods* for more information on how this works and potential problems that may result from improper use.

host_notification_commands

This directive is used to define a list of the *short names* of the *commands* used to notify the contact of a *host* problem or recovery. Multiple notification commands should be separated by commas. All notification commands are executed when the contact needs to be notified. The maximum amount of time that a notification command can run is controlled by the notification_timeout option.

service_notification_commands

This directive is used to define a list of the *short names* of the commands used to notify the contact of a *service* problem or recovery. Multiple notification commands should be separated by commas. All notification commands are executed when the contact needs to be notified. The maximum amount of time that a notification command can run is controlled by the notification_timeout option.

host_notification_options

This directive is used to define the host states for which notifications can be sent out to this contact. Valid options are a combination of one or more of the following:

- d = notify on DOWN host states
- u = notify on UNREACHABLE host states
- r = notify on host recoveries (UP states)
- f = notify when the host starts and stops *flapping*,
- s = send notifications when host or service *scheduled downtime* starts and ends. If you specify **n** (none) as an option, the contact will not receive any type of host notifications.

service_notification_options

This directive is used to define the service states for which notifications can be sent out to this contact. Valid options are a combination of one or more of the following:

- w = notify on WARNING service states
- u = notify on UNKNOWN service states
- c = notify on CRITICAL service states

- r = notify on service recoveries (OK states)
- f = notify when the service starts and stops *flapping*.
- n = (none) : the contact will not receive any type of service notifications.

min_business_impact

This directive is use to define the minimum business criticity level of a service/host the contact will be notified. Please see root_problems_and_impacts for more details.

- 0 = less important
- 1 = more important than 0
- 2 = more important than 1
- 3 = more important than 2
- 4 = more important than 3
- 5 = most important

8.16 Realm Definition

8.16.1 Description

The realms are a optional feature useful if the administrator want to divide it's resources like schedulers or pollers.

The Realm definition is optional. If no scheduler is defined, Shinken will "create" one for the user and will be the default one.

8.16.2 Definition Format

Variables in red are required, while those in black are optional. However, you need to supply at least one optional variable in each definition for it to be of much use.

define realm{	
realm_name	*realm_name*
realm_members	*realm_members*
default	*default*
}	

8.16.3 Example Definition:

```
define realm{
     realm_name     World
     realm_members  Europe,America,Asia
   default        0
}
```

8.16.4 Variable Descriptions

realm_name

This variable is used to identify the *short name* of the realm which the data is associated with.

realm_members

This directive is used to define the sub-realms of this realms.

default

This directive is used to define tis this realm is the default one (untagged host and satellites wil be put into it). The default value is *0*.

8.17 Arbiter Definition

8.17.1 Description

The Arbiter object is a way to defined Arbiter daemons that will manage all the configuration and all the architecture of shinken like distributed monitoring and high availability. It reads the configuration, cuts it into parts (N schedulers = N parts), and then send them to all others elements. It manages the high availability part : if an element dies, it re-routes the configuration managed by this falling element to a spare one. Its other role is to receive input from users (like external commands of shinken.cmd) and send them to other elements. There can be only one active arbiter in the architecture.

The Arbiter definition is optionnal. If no arbiter is defined, Shinken will "create" one for the user. There will be no high availability for the Arbiter (no spare), and will use the default port in the server where the deamon is launched.

8.17.2 Definition Format

Variables in red are required, while those in black are optional. However, you need to supply at least one optional variable in each definition for it to be of much use.

define arbiter{	
arbiter_name	*arbiter_name*
address	*dns name of ip address*
host_name	*hostname*
port	*port*
spare	//[0/1]//
modules	*modules*

timeout	*number of seconds to block the arbiter waiting for an answer*
data_timeout	*seconds to wait when sending data to another satellite(daemon)*
max_check_attempts	*number*
check_interval	*seconds to wait before issuing a new check*
}	

8.17.3 Example Definition:

```
define arbiter{
     arbiter_name       Main-arbiter
     address            node1.mydomain
     host_name          node1
     port               7770
     spare              0
     modules            module1,module2
}
```

8.17.4 Variable Descriptions

arbiter_name

This variable is used to identify the *short name* of the arbiter which the data is associated with.

address

This directive is used to define the adress from where the main arbier can reach this arbiter (that can be itself). This can be a DNS name or a IP adress.

host_name

This variable is used by the arbiters daemons to define with 'arbiter' object they are : all theses daemons in differents servers use the same configuration, so the only difference is the serveur name. This value must be equal to the name of the server (like with the hostname command). If none is defined, the arbiter daemon will put the name of the server where it's launched, but this will not be tolerated with more than one arbiter (because each daemons will think it's the master).

port

This directive is used to define the TCP port used bu the daemon. The default value is *7770*.

spare

This variable is used to define if the daemon matching this arbiter definition is a spare one of not. The default value is *0* (master).

modules

This variable is used to define all modules that the arbtier daemon matching this definition will load.

timeout

This variable defines how much time the arbiter will block waiting for the response of a inter-process ping (Pyro). 3 seconds by default. This operation will become non blocking when Python 2.4 and 2.5 is dropped in Shinken 1.4.

data_timeout

Data send timeout. When sending data to another process. 120 seconds by default.

max_check_attempts

If ping fails N or more, then the node is dead. 3 attempts by default.

check_interval

Ping node every N seconds. 60 seconds by default.

8.18 Scheduler Definition

8.18.1 Description

The Scheduler daemon is in charge of the scheduling checks, the analysis of results and follow up actions (like if a service is down, ask for a host check). They do not launch checks or notifications. They keep a queue of pending checks and notifications for other elements of the architecture (like pollers or reactionners). There can be many schedulers.

The Scheduler definition is optionnal. If no scheduler is defined, Shinken will "create" one for the user. There will be no high availability for it (no spare), and will use the default port in the server where the deamon is launched.

8.18.2 Definition Format

Variables in red are required, while those in black are optional. However, you need to supply at least one optional variable in each definition for it to be of much use.

define scheduler{	
scheduler_name	*scheduler_name*
address	*dns name of ip address*
port	*port*
spare	//[0/1]//
realm	*realm name*
modules	*modules*
}	

8.18.3 Example Definition:

```
define scheduler{
     scheduler_name     Europe-scheduler
     address            node1.mydomain
     port               7770
     spare              0
     realm              Europe

     ## Optional
     spare               0   ; 1 = is a spare, 0 = is not a spare
     weight              1   ; Some schedulers can manage more hosts than others
     timeout             3   ; Ping timeout
     data_timeout        120 ; Data send timeout
     max_check_attempts  3   ; If ping fails N or more, then the node is dead
     check_interval      60  ; Ping node every minutes
     modules            PickleRetention

     # Skip initial broks creation for faster boot time. Experimental feature
     # which is not stable.
     skip_initial_broks  0
     # In NATted environments, you declare each satellite ip[:port] as seen by
     # *this* scheduler (if port not set, the port declared by satellite itself
     # is used)
     #satellitemap    poller-1=1.2.3.4:1772, reactionner-1=1.2.3.5:1773, ...
}
```

}

8.18.4 Variable Descriptions

scheduler_name

This variable is used to identify the *short name* of the scheduler which the data is associated with.

address

This directive is used to define the adress from where the main arbier can reach this scheduler. This can be a DNS name or a IP adress.

port

This directive is used to define the TCP port used bu the daemon. The default value is *7768*.

spare

This variable is used to define if the scheduler must be managed as a spare one (will take the conf only if a master failed). The default value is *0* (master).

realm

This variable is used to define the *realm* where the scheduler will be put. If none is selected, it will be assigned to the default one.

modules

This variable is used to define all modules that the scheduler will load.

8.19 Poller Definition

8.19.1 Description

The Poller object is a way to the Arbiter daemons to talk with a scheduler and give it hosts to manage. They are in charge of launching plugins as requested by schedulers. When the check is finished they return the result to the schedulers. There can be many pollers.

The Poller definition is optionnal. If no poller is defined, Shinken will "create" one for the user. There will be no high availability for it (no spare), and will use the default port in the server where the deamon is launched.

8.19.2 Definition Format

Variables in red are required, while those in black are optional. However, you need to supply at least one optional variable in each definition for it to be of much use.

define poller{	
poller_name	*poller_name*
address	*dns name of ip address*
port	*port*
spare	//[0/1]//
realm	*realm name*
manage_sub_realms	*[0,1]*
poller_tags	*poller_tags*
modules	*modules*
}	

8.19.3 Example Definition:

```
define poller{
     poller_name        Europe-poller
     address            node1.mydomain
     port               7771
     spare              0


     # Optional configurations
     manage_sub_realms  0
     poller_tags        DMZ, Another-DMZ
     modules            module1,module2
```

```
    realm              Europe
    min_workers         0   ; Starts with N processes (0 = 1 per CPU)
    max_workers         0   ; No more than N processes (0 = 1 per CPU)
    processes_by_worker 256 ; Each worker manages N checks
    polling_interval    1   ; Get jobs from schedulers each 1 minute
}
```

8.19.4 Variable Descriptions

poller_name

This variable is used to identify the *short name* of the poller which the data is associated with.

address

This directive is used to define the adress from where the main arbier can reach this poller. This can be a DNS name or a IP adress.

port

This directive is used to define the TCP port used bu the daemon. The default value is *7771*.

spare

This variable is used to define if the poller must be managed as a spare one (will take the conf only if a master failed). The default value is *0* (master).

realm

This variable is used to define the *realm* where the poller will be put. If none is selected, it will be assigned to the default one.

manage_sub_realms

This variable is used to define if the poller will take jobs from scheduler from the sub-realms of it's realm. The default value is *0*.

poller_tags

This variable is used to define the checks the poller can take. If no poller_tags is defined, poller will take all untagued checks. If at least one tag is defined, it will take only the checks that are also taggued like it.

By default, there is no poller_tag, so poller can take all untagued checks (default).

modules

This variable is used to define all modules that the scheduler will load.

8.20 Reactionner Definition

8.20.1 Description

The Reactionner daemon is in charge of notifications and launching event_handlers. There can be more than one Reactionner.

8.20.2 Definition Format

Variables in red are required, while those in black are optional. However, you need to supply at least one optional variable in each definition for it to be of much use.

define reactionner{	
reactionner_name	*reactionner_name*
address	*dns name of ip address*
port	*port*
spare	//[0/1]//
realm	*realm name*
manage_sub_realms	*[0,1]*
modules	*modules*
}	

8.20.3 Example Definition:

```
define reactionner{
     reactionner_name   Main-reactionner
     address            node1.mydomain
     port               7769
     spare              0
     realm              All

     ## Optionnal
     manage_sub_realms   0   ; Does it take jobs from schedulers of sub-Realms?
     min_workers         1   ; Starts with N processes (0 = 1 per CPU)
     max_workers         15  ; No more than N processes (0 = 1 per CPU)
     polling_interval    1   ; Get jobs from schedulers each 1 second
     timeout             3   ; Ping timeout
     data_timeout        120 ; Data send timeout
     max_check_attempts  3   ; If ping fails N or more, then the node is dead
     check_interval      60  ; Ping node every minutes
     reactionner_tags   tag1
     modules            module1,module2
}
```

8.20.4 Variable Descriptions

reactionner_name

This variable is used to identify the *short name* of the reactionner which the data is associated with.

address

This directive is used to define the adress from where the main arbier can reach this reactionner. This can be a DNS name or a IP adress.

port

This directive is used to define the TCP port used bu the daemon. The default value is *7772*.

spare

This variable is used to define if the reactionner must be managed as a spare one (will take the conf only if a master failed). The default value is *0* (master).

realm

This variable is used to define the *realm* where the reactionner will be put. If none is selected, it will be assigned to the default one.

manage_sub_realms

This variable is used to define if the poller will take jobs from scheduler from the sub-realms of it's realm. The default value is *1*.

modules

This variable is used to define all modules that the scheduler will load.

reactionner_tags

This variable is used to define the checks the reactionner can take. If no reactionner_tags is defined, reactionner will take all untagued notifications and event handlers. If at least one tag is defined, it will take only the checks that are also taggued like it.

By default, there is no reactionner_tag, so reactionner can take all untagued notification/event handlers (default).

8.21 Broker Definition

8.21.1 Description

The Broker daemon provides access to Shinken internal data. Its role is to get data from schedulers (like status and logs) and manage them. The management is done by modules. Many different modules exists : export to graphite, export to syslog, export into ndo database (MySQL and Oracle backend), service-perfdata export, couchdb export and more. To configure modules, consult the *broker module definitions*.

The Broker definition is optional.

8.21.2 Definition Format

Variables in red are required, while those in black are optional. However, you need to supply at least one optional variable in each definition for it to be of much use.

define broker{	
broker_name	*broker_name*
address	*dns name of ip address*
port	*port*
spare	//[0/1]//
realm	*realm name*
manage_sub_realms	*[0,1]*
modules	*modules*
}	

8.21.3 Example Definition:

```
define broker{
     broker_name        broker-1
     address            node1.mydomain
     port               7772
     spare              0
     realm              All
     ## Optional
```

```
    manage_arbiters     1
    manage_sub_realms   1
    timeout             3   ; Ping timeout
    data_timeout        120 ; Data send timeout
    max_check_attempts  3   ; If ping fails N or more, then the node is dead
    check_interval      60  ; Ping node every minutes              manage_sub_realms  1
    modules             Livestatus,Simple-log,WebUI
}
```

8.21.4 Variable Descriptions

broker_name

This variable is used to identify the *short name* of the broker which the data is associated with.

address

This directive is used to define the adress from where the main arbier can reach this broker. This can be a DNS name or a IP adress.

port

This directive is used to define the TCP port used bu the daemon. The default value is *7772*.

spare

This variable is used to define if the broker must be managed as a spare one (will take the conf only if a master failed). The default value is *0* (master).

realm

This variable is used to define the *realm* where the broker will be put. If none is selected, it will be assigned to the default one.

manage_arbiters

Take data from Arbiter. There should be only one broker for the arbiter.

manage_sub_realms

This variable is used to define if the broker will take jobs from scheduler from the sub-realms of it's realm. The default value is *1*.

modules

This variable is used to define all modules that the broker will load. The main goal ofthe Broker is to give status to theses modules.

CHAPTER 9

Shinken Architecture

9.1 Arbiter supervision of Shinken processes

9.1.1 Introduction

Nobody is perfect, nor are OSes. A server can fail, and so does the network. That's why you can (should) define multiple processes as well as spares in the Shinken architecture.

9.1.2 Supervision method

The Arbiter daemon constantly checks that every one is alive. If a node is declared to be dead, the Arbiter will send the configuration of this node to a spare one. Other satellites get the address of this new node so they can change their connections.

The case where the daemon was still alive but that it was just a network interruption is also managed: There will be 2 nodes with the same configuration, but the Arbiter will ask one (the old one) to become inactive.

The spare node that has become active will not preemptively fail back to the original node.

9.1.3 Adjusting timers for large configurations

Supervision parameters need to be adjusted with extra large configurations that may need more time to startup or process data. The arbiter periodically sends a verification to each process to see if it is alive. The timeouts and retry for this can be adjusted.

Timeout should be left at 3 seconds, even for large configuration. Health checks are synchronous. This is due to compatibility with Python 2.4. There will eventually become asynchronous as Python 2.6 support is dropped in the future. Retries should be multiplied as need. Data timeout should be left as is.

Important: TODO Put all of the arbiter supervisory parameters with formulas and examples.

9.1.4 Diagrams

It can be shown in the following diagram:

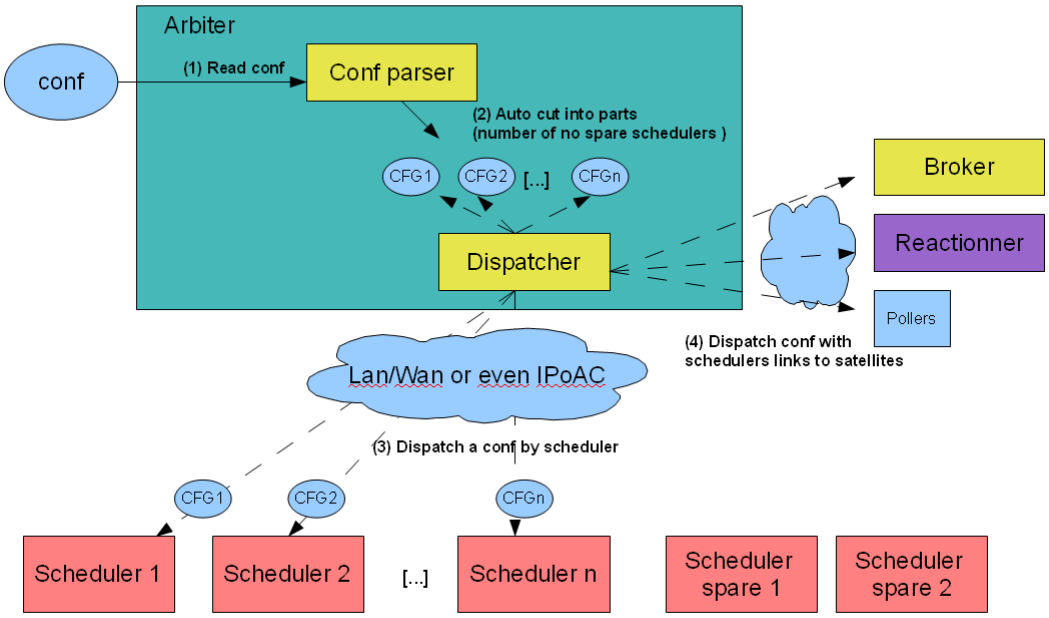
Arbiter
conf
(1) Read conf
Conf parser
(2) Auto cut into parts
(number of no spare schedulers)
CFG1
CFG2
[...]
CFGn
Dispatcher
Broker
Reactionner
Pollers
(4) Dispatch conf with
schedulers links to satellites
Lan/Wan or even IPoAC
(3) Dispatch a conf by scheduler
CFG1
CFG2
CFGn
Scheduler 1
Scheduler 2
[...]
Scheduler n
Scheduler
spare 1
Scheduler
spare 2

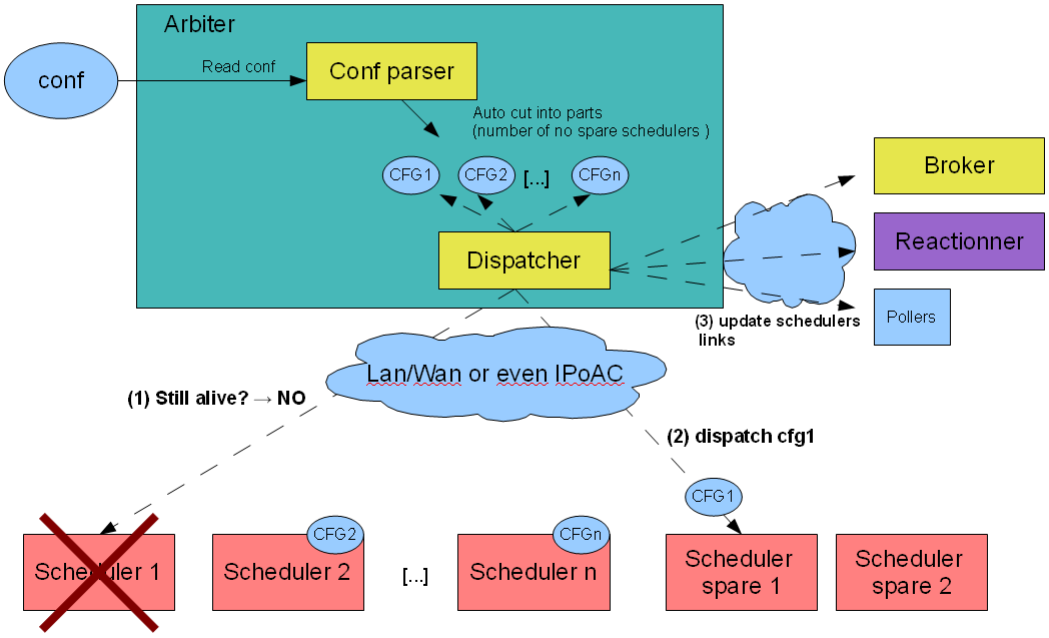
Arbiter
conf
Read conf
Conf parser
Auto cut into parts
(number of no spare schedulers)
CFG1
CFG2
[...]
CFGn
Dispatcher
Broker
Reactionner
Pollers
(3) update schedulers
links
Lan/Wan or even IPoAC
(1) Still alive? → NO
(2) dispatch cfg1
CFG1
CFG2
CFGn
Scheduler 1
Scheduler 2
[...]
Scheduler n
Scheduler
spare 1
Scheduler
spare 2

9.2 Advanced architectures

Shinken has got some cool features in term of configuration management or performance, but it's true added value is in its architecture. It's been designed to achieve easy distributed architecture and high availability.

9.2.1 = Distributed architecture

The load balancing feature is very easy to obtain with Shinken. If I say that the project's name comes from it you should believe me :-)

In fact the load is present in 2 major places:

- pollers: they launch checks, they use a lot of resources
- schedulers: they schedule, potentially lots of checks

For both, a limit of 150000 checks/5min is a reasonable goal on an average server(4 cores@3Ghz). But remember that is can be multiplied as much as you wish, just by adding another server.

There are 2 cases:

- checks that ask for a lot of performance (perl or shell scripts for example)
- a lot of scheduling (> 150000 checks in 5 minutes).

In the first case, you need to add more pollers. In the second, you need to add more schedulers. In this last case, you should also add more pollers (more launch need more pollers) but that's not compulsory.

But I already ear you asking "How to add new satellites?". That's very simple: You start by installing the application on a new server (don't forget the sinken user + application files). Let say that this new server is called server-2 and has the IP 192.168.0.2 (remember that the "master" is called server-1 with 192.168.0.1 as IP).

Now you need to launch the scheduler and pollers (or just one of them if you want):

```
/etc/init.d/shinken-scheduler start
/etc/init.d/shinken-poller start
```

It looks like the launch in the master server? Yes, it's the same :-)

Here you just launch the daemons, now you need to declare them in the shinken-specific.cfg file on the master server (the one with the arbiter daemon). You need to add new entries for these satellites:

```
define scheduler{
```

```
     scheduler_name   scheduler-2
     address  192.168.0.2
     port     7768
     spare    0
     }

define poller{
```

```
     poller_name     poller-2
     address  192.168.0.2
     port     7771
     spare    0
}
```

The differences with scheduler-1 and poller-1 are just the names and the address. Yep, that's all :-)

Now you can restart the arbiter, and you're done, the hosts will be distributed in both schedulers, and the checks will be distributed to all pollers. Congrats :)

High availability architecture

Ok, a server can crash or a network can go down. The high availability is not a useless feature, you know?

With shinken, making a high availability architecture is very easy. Just as easy as the load balancing feature :)

You saw how to add new scheduler/poller satellites. For the HA it's quite the same. You just need to add new satellites in the same way you just did, and define them as "spares". You can (should) do the same for all the satellites (a new arbiter, reactionner and broker) for a whole HA architecture.

We keep the load balancing previous installation and we add a new server (if you do not need load balancing, just take the previous server). So like the previous case, you need to install the daemons and launch them.

```
/etc/init.d/shinken-scheduler start
/etc/init.d/shinken-poller start
```

Nothing new here.

And we need to declare the new satellites in the shinken-specific.cfg file near the arbiter:

```
define scheduler{
```

```
     scheduler_name   scheduler-3
     address  192.168.0.3
     port     7768
     spare    1
     }

define poller{
```

```
     poller_name     poller-3
     address  192.168.0.3
     port     7771
     spare    1
}
```

Do you see a difference here? Aside from the name and address, there is a new parameter: spare. From the daemon point of view, they do not know if they are a spare or not. That's the arbiter that says what they are.

You just need to restart the arbiter and you've got HA for the schedulers/pollers :)

Really?

Yes indeed, that's all. Until one of the scheduler/poller fall, these two new daemons will just wait. If anyone falls, the spare daemon will take the configuration and start to work.

You should do the same for arbiter, reactionner and broker. Just install them in the server-3 and declare them in the shinken-specific.cfg file with a spare parameter. Now you've got a full HA architecture (and with load balancing if you keep the server-2 :)).

Note: Here you have high availability, but if a scheduler dies, the spare takes the configuration, but not the saved states. So it will have to reschedule all checks, and current states will be PENDING. To avoid this, you can link distributed retention modules <distributed retention modules> such as memcache to your schedulers

Mixed Architecture (poller GNU/Linux and Windows or LAN/DMZ)

There can be as many pollers as you want. And Shinken runs under a lot of systems, like GNU/Linux and Windows. It could be useful to make windows hosts checks by a windows pollers (by a server IN the domain), and all the others by a GNU/Linux one.

And in fact you can, and again it's quite easy :) All pollers connect to all schedulers, so we must have a way to distinguish 'windows' checks from 'gnu/linux' ones.

The poller_tag/poller_tags parameter is useful here. It can be applied on the following objects:

- pollers
- commands
- services
- hosts

It's quite simple: you 'tag' objects, and the pollers have got tags too. You've got an implicit inheritance between hosts->services->commands. If a command doesn't have a poller_tag, it will take the one from the service. And if this service doesn't have one neither, it will take the tag from its host.

Let take an example with a 'windows' tag:

```
define command{
```

```
 command_name
 command_line   c:\shinken\libexec\check_wmi.exe -H $HOSTADRESS$ -r $ARG1$
 poller_tag     Windows
}

define poller{
```

```
 poller_name  poller-windows
 address      192.168.0.4
 port     7771
 spare    0
 poller_tags  Windows,DMZ
}
```

And the magic is here: all checks launched with this command will be taken by the poller-windows (or another that has such a tag). A poller with no tags will only take 'untagged' commands.

It also works with a LAN/DMZ network. If you do not want to open all monitoring ports from the LAN to the DMZ server, you just need to install a poller with the 'DMZ' tag in the DMZ and then add it to all hosts (or services) in the DMZ. They will be taken by this poller and you just need to open the port to this poller from the LAN. Your network admins will be happier :)

```
define host{
```

```
 host_name  server-DMZ-1
 [...]
 poller_tag DMZ
 [...]
}

define service{
```

```
 service_description  CPU
 host_name  server-DMZ-2
```

```
 [...]
 poller_tag DMZ
 [...]
}
```

And that's all :)

Multi customers and/or sites: REALMS

The shinken's architecture like we saw allows us to have a unique administration and data location. All pollers the hosts are cut and sent to schedulers, and the pollers take jobs from all schedulers. Every one is happy.

Every one? In fact no. If an administrator got a continental distributed architecture he can have serious problems. If the architecture is common to multile customers network, a customer A scheduler can have a customer B poller that asks him jobs. It's not a good solution. Even with distributed network, distant pollers should not ask jobs to schedulers in the other continent, it's not network efficient.

That is where the site/customers management is useful. In Shinken, it's managed by the **realms**.

A realm is a group of resources that will manage hosts or hostgroups. Such a link will be unique: a host cannot be in multiple realms. If you put an hostgroup in a realm, all hosts in this group will be in the realm (unless a host already has the realm set, the host value will be taken).

A realm is:

- at least a scheduler
- at least a poller
- can have a reactionner
- can have a broker

In a realm, all realm pollers will take all realm schedulers jobs.

Important: Very important: there is only ONE arbiter (and a spare of couse) for ALL realms. The arbiter manages all realms and all that is inside.

Sub-realms

A realm can have sub-realms. It doesnt change anything for schedulers, but it can be useful for other satellites and spares. Reactionners and brokers are linked to a realm, but they can take jobs from all sub-realms too. This way you can have less reactionners and brokers (like we soon will see).

The fact that reactionners/brokers (and in fact pollers too) can take jobs from sub-schedulers is decided by the presence of the manage_sub_realms parameter. For pollers the default value is 0, but it's 1 for reactionners/brokers.

Important: WARNING: having multiple brokers for one scheduler is not a good idea: after the information is send, it's deleted from the scheduler, so each brokers will only got partial data!

An example ?

To make it simple: you put hosts and/or hostgroups in a realm. This last one is to be considered as a resources pool. You don't need to touch the host/hostgroup definition if you need more/less performances in the realm or if you want to add a new satellites (a new reactionner for example).

Realms are a way to manage resources. They are the smaller clouds in your global cloud infrastructure :)

If you do not need this feature, that's not a problem, it's optional. There will be a default realm created and every one will be put into.

It's the same for hosts that don't have a realm configured: they will be put in the realm that has the "default" parameter.

Picture example

Diagrams are good :)

Let's take two examples of distributed architectures around the world. In the first case, the administrator don't want to share resources between realms. They are distinct. In the second, the reactionners and brokers are shared with all realms (so all notifications are send from a unique place, and so is all data).

Here is the isolated one:

And a more common way of sharing reactionner/broker:

Like you can see, all elements are in a unique realm. That's the sub-realm functionality used for reactionner/broker.

Configuration of the realms

Here is the configuration for the shared architecture:

```
define realm {
```

```
 realm_name       All
 realm_members    Europe,US,Asia
 default          1    ;Is the default realm. Should be unique!
}
define realm{
```

```
 realm_name       Europe
 realm_members    Paris   ;This realm is IN Europe
}
```

An now the satellites:

```
define scheduler{
```

```
 scheduler_name       scheduler_Paris
 realm                Paris             ;It will only manage Paris hosts
}
define reactionner{
```

```
 reactionner_name     reactionner-master
 realm                All                ;Will reach ALL schedulers
}
```

And in host/hostgroup definition:

```
define host{
```

```
 host_name          server-paris
 realm              Paris         ;Will be put in the Paris realm
 [...]
}

define hostgroups{
```

```
 hostgroup_name                linux-servers
 alias                         Linux Servers
 members                       srv1,srv2
 realm                         Europe       ;Will be put in the Europe realm
}
```

9.3 How are commands and configurations managed in Shinken

Let's take a look at how the dispatching is managed.

Shinken uses different daemons: each one has it's own task. The global master is the Arbiter: it reads the configuration, divides it into parts and sends the parts to various Shinken daemons. It also looks at which daemon are alive: if one dies, it will give the configuration of the dead one to another daemon so that it can replace it.

9.3.1 Configuration dispatching

It looks like this:

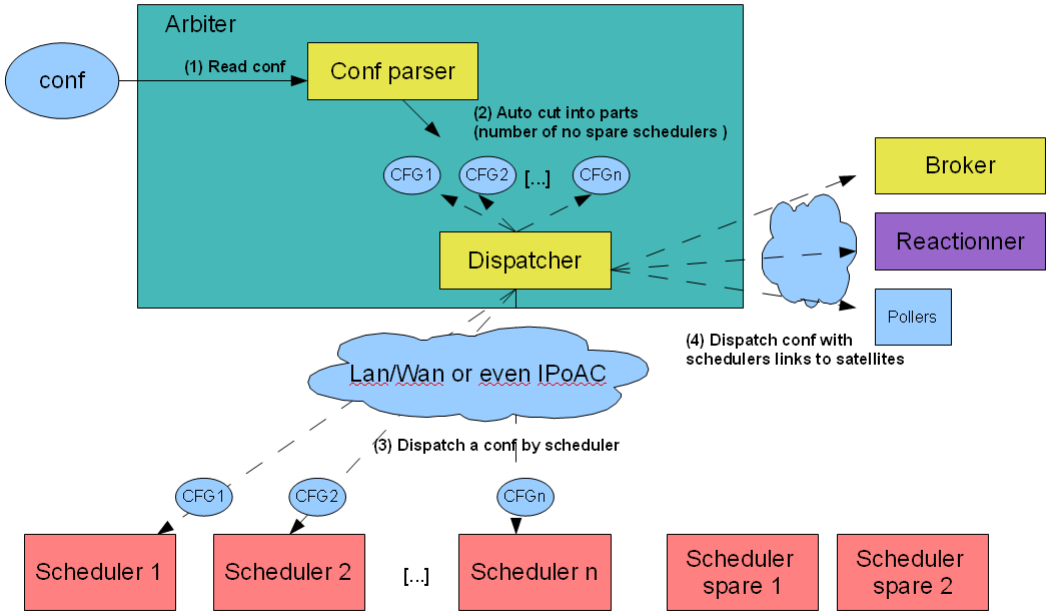

9.3.2 Configuration changes on running systems

Once the configuration is being dispatched to a Shinken process by the Arbiter, this causes the process (ex. Scheduler) to stop and reload its configuration. Thus for small configurations, the monitoring gap, where no monitoring is being done, is of an inconsequential duration. However, as the number of services rises above 10K and as the complexity of the configuration grows, the monitoring gap will become noticeable to the order of minutes. This gap will impact the type of SLA the monitoring solution can meet. .. important:

```
The 1.2 release is mandatory for anyone using more than 10K services as it includes improvements addr
```

9.3.3 External commands dispatching

Method 1 - Via the Arbiter daemon

The user can send external commands to the system to raise a downtime or ask for passive checks. The commands should be sent to the only daemon that orchestrates everything in the system (the only one the user should know about): the Arbiter. It gives him the external commands (in a name pipe from now) and the Arbiter sends it to whoever needs it (just one scheduler or all of them).

It looks like this:

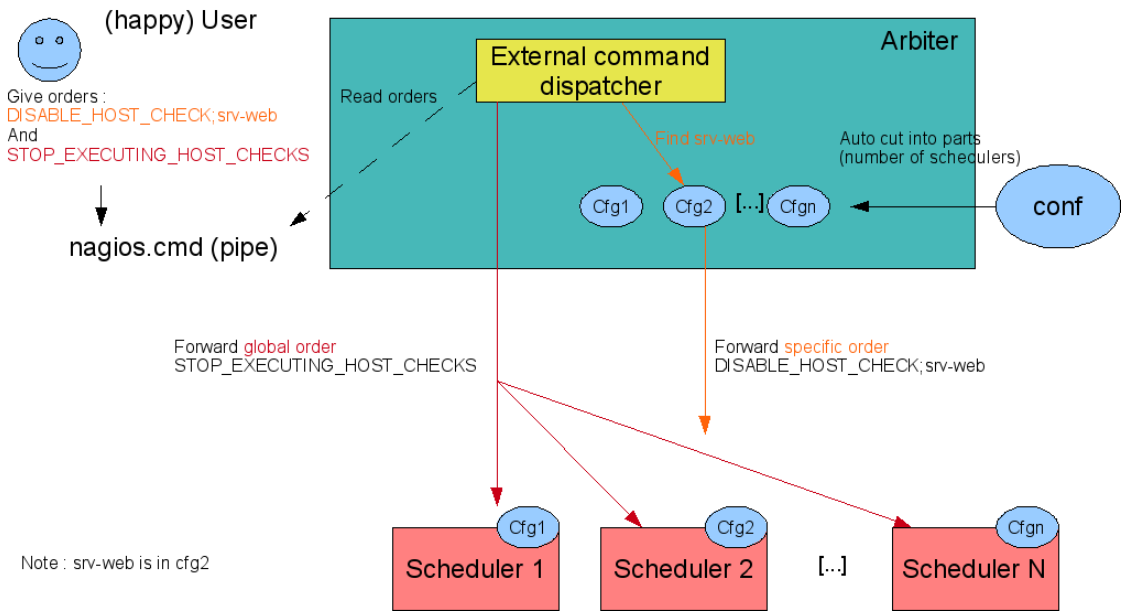

Method 2 - Via the Livestatus API

The Livestatus API is a Broker daemon module. It listens on the network for data requests or external commands. Commands can be authenticated. Commands can be submitted via a Python interface or a JSON interface. Consult the MK Livestatus documentation for the supported parameters.

9.4 Problems and impacts correlation management

9.4.1 What is this correlation ?

The main role of this feature is to allow users to have the same correlation views in the console than they got in the notifications.

From now, users won"t get notified if there was a dependency problem or example (a host in DOWN make the service notifications to not be send for example). But in the console, we still got some green and red indicators: the scheduler waited for actual checks to put the elements in a UNKNOWN or UNREACHABLE state when he already know that there was a dependency problem.

Now it"s smart enough to put such states to elements that we know the check will fail. An example?

Imagine such a parent relations between hosts:

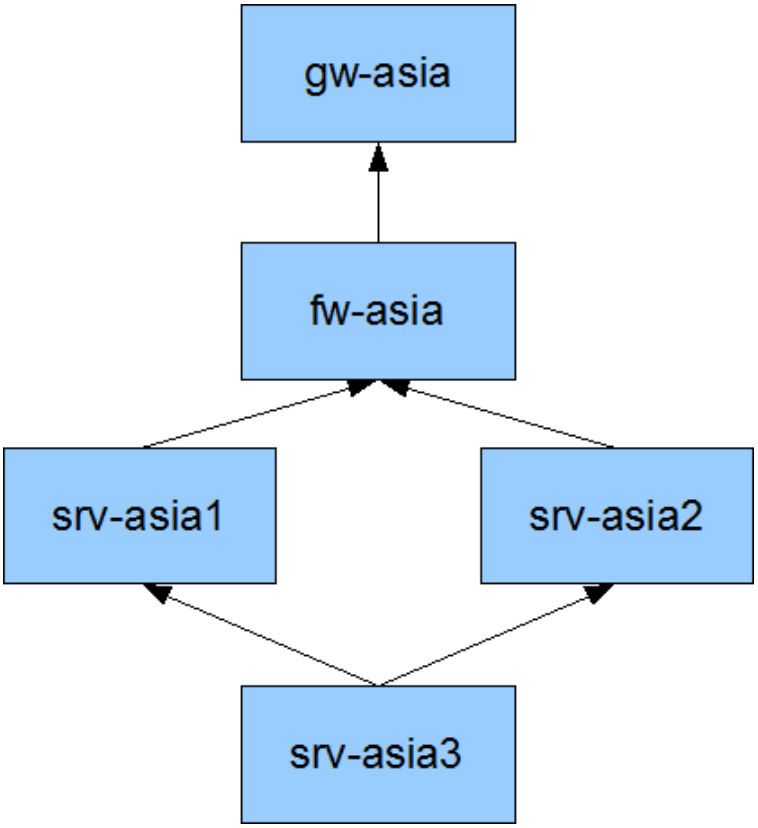

If gw is DOWN, all checks for others elements will put UNREACHABLE state. But if the fw and servers are checks 5 minutes later, during this period, the console will still have green indicators for them. And are they really green? No. We know that future checks will put them in errors. That why the problems/impacts feature do: when the gateway is set in HARD/DOWN, it apply a UNREACHABLE (and UNKNOWN for services) states for others elements below. So the administrators in front of his desk saw directly that there is a problem, and what are the elements impacted.

It"s important to see that such state change do not interfere with the HARD/SOFT logic: it"s just a state change for console, but it"s not taken into account as a checks attempt.

Here gateway is already in DOWN/HARD. We can see that all servers do not have an output: they are not already checked, but we already set the UNREACHABLE state. When they will be checks, there will be an output and they will keep this state.

9.4.2 How to enable it?

It's quite easy, all you need is to enable the parameter

```
enable_problem_impacts_states_change=1
```

See the page *Main advanced configuration* <configuringshinken-configmain-advanced> for more information about it.

9.4.3 Dynamic Business Impact

There is a good thing about problems and impacts when you do not identify a parent devices Business Impact: your problem will dynamically inherit the maximum business impact of the failed child!

Let take an example: you have a switch with different children, one is a devel environment with a low business impact (0 or 1) and one with a huge business impact (4 or 5). The network administrator has set SMS notification at night but only for HUGE business impacts (min_criticity=4 in the contact definition for example).

It's important to say that the switch DOES NOT HAVE ITS OWN BUSINESS IMPACT DEFINED! A switch is there to server applications, the only business impact it gets is the child hosts and services that are connected to it!

There are 2 nights:

- the first one, the switch got a problem, and only the devel environment is impacted. The switch will inherit the maximum impact of its impacts (or it own value if it's higher, it's 3 by default for all elements). Here the devel impact is 0, the switch one is 3, so its impact will stay at 3. It's slower than the contact value, so the notification will not be send, the admin will be able to sleep :)
- the second night, the switch got a problem, but this time it impacts the production environment! This time, the computed impact is set at 5 (the one of the max impact, here the production application), so it's higher than the min_criticity of the contact, so the notification is send. The admin is awaken, and can solve this problem before too many users are impacted :)

9.5 Problems and impacts correlation management

9.5.1 What is this correlation ?

The main role of this feature is to allow users to have the same correlation views in the console than they got in the notifications.

From now, users won"t get notified if there was a dependency problem or example (a host in DOWN make the service notifications to not be send for example). But in the console, we still got some green and red indicators: the scheduler waited for actual checks to put the elements in a UNKNOWN or UNREACHABLE state when he already know that there was a dependency problem.

Now it"s smart enough to put such states to elements that we know the check will fail. An example?

Imagine such a parent relations between hosts:

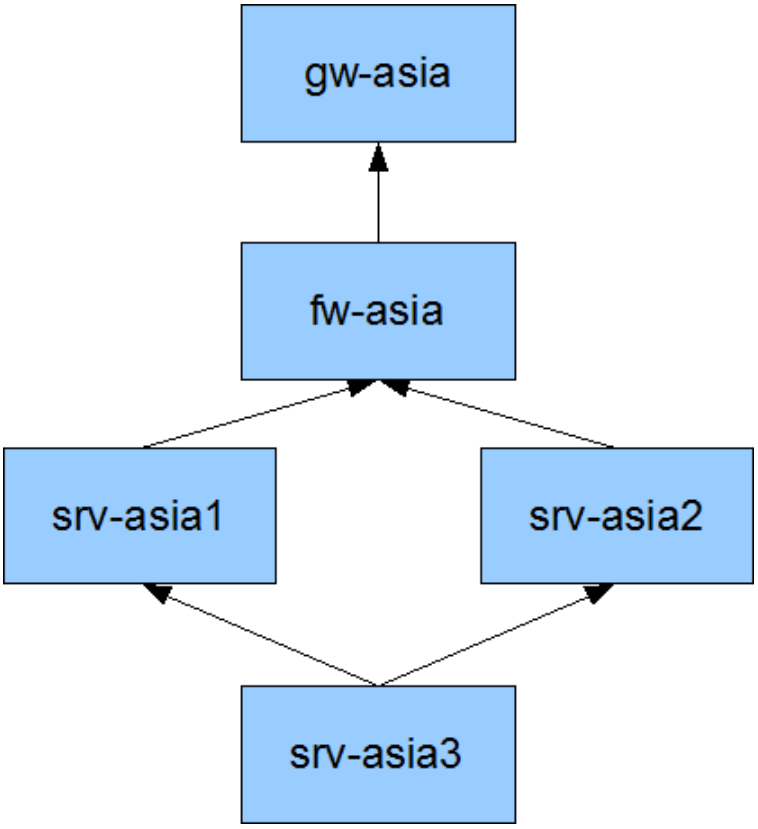

If gw is DOWN, all checks for others elements will put UNREACHABLE state. But if the fw and servers are checks 5 minutes later, during this period, the console will still have green indicators for them. And are they really green? No. We know that future checks will put them in errors. That why the problems/impacts feature do: when the gateway is set in HARD/DOWN, it apply a UNREACHABLE (and UNKNOWN for services) states for others elements below. So the administrators in front of his desk saw directly that there is a problem, and what are the elements impacted.

It"s important to see that such state change do not interfere with the HARD/SOFT logic: it"s just a state change for console, but it"s not taken into account as a checks attempt.

Here gateway is already in DOWN/HARD. We can see that all servers do not have an output: they are not already checked, but we already set the UNREACHABLE state. When they will be checks, there will be an output and they will keep this state.

9.5.2 How to enable it?

It's quite easy, all you need is to enable the parameter

```
enable_problem_impacts_states_change=1
```

See the page *Main advanced configuration* <configuringshinken-configmain-advanced> for more information about it.

9.5.3 Dynamic Business Impact

There is a good thing about problems and impacts when you do not identify a parent devices Business Impact: your problem will dynamically inherit the maximum bushiness impact of the failed child!

Let take an example: you have a switch with different children, one is a devel environment with a low business impact (0 or 1) and one with a huge business impact (4 or 5). The network administrator has set SMS notification at night but only for HUGE business impacts (min_criticity=4 in the contact definition for example).

It's important to say that the switch DOES NOT HAVE ITS OWN BUSINESS IMPACT DEFINED! A switch is there to server applications, the only business impact it gets is the child hosts and services that are connected to it!

There are 2 nights:

- the first one, the switch got a problem, and only the devel environment is impacted. The switch will inherit the maximum business impact of its impacts (or it own value if it's higher, it's 3 by default for all elements). Here the devel impact is 0, the switch one is 3, so its impact will stay at 3. It's slower than the contact value, so the notification will not be send, the admin will be able to sleep :)
- the second night, the switch got a problem, but this time it impacts the production environment! This time, the computed impact is set at 5 (the one of the max impact, here the production application), so it's higher than the min_criticity of the contact, so the notification is send. The admin is awaken, and can solve this problem before too many users are impacted :)

9.6 Shinken Architecture

9.6.1 Summary

Shinken"s architecture has been designed according to the Unix Way: one tool, one task. Shinken has an architecture where each part is isolated and connects to the others via standard interfaces. Shinken is based on the Python Pyro remote objects library. Which makes building a highly available or distributed monitoring architectures quite easy.

- Shinken gets data IN * passively * actively * Networked API
- Shinken acts on the data * Correlation * Event suppression * Event handlers * Adding new poller daemons * Runtime interaction
- Shinken gets data OUT * Networked API * Notifications * Logging * Web/Mobile Frontend (via API and Native WebUI) * Metrics databases
- Shinken manages configurations * Discovery manager SkonfUI * Multi-level discovery engine * Configuration Packs (commands, config templates, graph templates, etc.) * Text file management via configuration engines (cfengine, chef, puppet, salt)

9.6.2 Shinken innovative features

Learn more about the innovative features of Shinken.

9.6.3 Shinken data acquisition for monitoring

Shinken needs plugins to actually gather data. There exists **thousands** of plugins for every conceivable application. Shinken packages the configurations necessary to use common plugins in *Packs*. Plugins themselves need to be installed by the administrator of the monitoring solution(Shinken will install some common ones for you). This is a great strength and flexibility of Shinken, but also an administrative responsibility to download and install the necessary plugins.

9.6.4 Architecture diagram with all daemons illustrated

9.6.5 Shinken Daemon roles

- **:ref:'Arbiter <arbiter>'**: The arbiter daemon reads the configuration, divides it into parts (N schedulers = N parts), and distributes them to the appropriate Shinken daemons. Additionally, it manages the high availability features: if a particular daemon dies, it re-routes the configuration managed by this failed daemon to the configured spare. Finally, it can receive input from users (such as external commands from nagios.cmd) or passive check results and routes them to the appropriate daemon. Passive check results are forwarded to the Scheduler responsible for the check. There can only be one active arbiter with other arbiters acting as hot standby spares in the architecture. * Modules for data collection: *NSCA*, *TSCA*, *Ws_arbiter* (web service) * Modules for configuration data storage: MongoDB * Modules for status retention: PickleRententionArbiter * Modules for configuration import: MySQLImport, *GLPI*, *Landscape(Ubuntu)* * Modules for configuration modification: *vmware autolinking*, *IP_Tag*, and other task specific modules
- **:ref:'Scheduler <scheduler>'**: The scheduler daemon manages the dispatching of checks and actions to the poller and reactionner daemons respectively. The scheduler daemon is also responsible for processing the check result queue, analyzing the results, doing correlation and following up actions accordingly (if a service is down, ask for a host check). It does not launch checks or notifications. It just keeps a queue of pending checks and notifications for other daemons of the architecture (like pollers or reactionners). This permits distributing load equally across many pollers. There can be many schedulers for load-balancing or hot standby roles. *Status persistence is achieved using a retention module*. * Modules for status retention: pickle, nagios, memcache, redis and MongoDB are available.
- **:ref:'Poller <poller>'**: The poller daemon launches check plugins as requested by schedulers. When the check is finished it returns the result to the schedulers. Pollers can be tagged for specialized checks (ex. Windows versus Unix, customer A versus customer B, DMZ) There can be many pollers for load-balancing or hot standby spare roles. * Module for data acquisition: *NRPE Module* * Module for data acquisition: CommandFile (Used for check_mk integration which depends on the nagios.cmd named pipe) * Module for data acquisition: *Snmp-Booster* (NEW)
- **:ref:'Reactionner <reactionner>'**: The reactionner daemon issues notifications and launches event_handlers. This centralizes communication channels with external systems in order to simplify SMTP authorizations or RSS feed sources (only one for all hosts/services). There can be many reactionners for load-balancing and spare roles * Module for external communications: *AndroidSMS*
- **:ref:'Broker <broker>'**: The broker daemon exports and manages data from schedulers. The broker uses modules exclusively to get the job done. The main method of interacting with Shinken is through the Livestatus API. Learn how to configure the Broker modules. * Modules for the Livestatus API - live state, status retention and history: SQLite (default), MongoDB (experimental) * Module for centralizing Shinken logs: Simple-log (flat file) * Modules for data retention: Pickle , ToNdodb_Mysql, ToNdodb_Oracle, ~~couchdb~~ * Modules for exporting data: Graphite-Perfdata, NPCDMOD(PNP4Nagios), raw_tcp(Splunk), Syslog * Modules for the Shinken WebUI: GRAPHITE_UI, PNP_UI. Trending and data visualization. * Modules for compatibility/migration: Service-Perfdata, Host-Perfdata and Status-Dat
- **Receiver** (optional): The receiver daemon receives passive check data and serves as a distributed command buffer. There can be many receivers for load-balancing and hot standby spare roles. The receiver can also use modules to accept data from different protocols. Anyone serious about using passive check results should use a receiver to ensure that check data does not go through the Arbiter (which may be busy doing administrative tasks) and is forwarded directly to the appropriate Scheduler daemon(s). * Module for passive data collection: *NSCA*, *TSCA*, *Ws_arbiter (web service)*

Tip: The various daemons can be run on a single server for small deployments or split on different hardware for larger deployments as performance or availability requirements dictate. For larger deployments, running multiple Schedulers is recommended, even if they are on the same server. Consult planning a large scale Shinken deployment for more information.

9.6.6 Learn more about the Shinken Distributed Architecture

The Shinken distributed architecture, more features explained.

- Smart and automatic load balancing
- High availability
- Specialized Pollers
- Advanced architectures: Realms

If you are just starting out, you can continue on with the next tutorial, which will help you configure a web front-end.

9.6.7 Planning a large scale Shinken deployment

If you wish to plan a large scale installation of Shinken, you can consult the Scaling Shinken reference.

This is essential to avoid making time consuming mistakes and aggravation.

CHAPTER 10

Troubleshooting

10.1 FAQ - Shinken troubleshooting

10.1.1 FAQ Summary =

Shinken users, developers, administrators possess a body of knowledge that usually provides a quick path to problem resolutions. The Frequently Asked Questions questions are compiled from user questions and issues developers may run into.

Have you consulted at all the *resources available for users and developers*.

__Before posting a question to the forum:__

- Read the through the Getting Started tutorials
- Search the documentation wiki
- Use this FAQ
- Bonus: Update this FAQ if you found the answer and think it would benefit someone else

Doing this will improve the quality of the answers and your own expertise.

Frequently asked questions

- How to set my daemons in debug mode to review the logs?
- I am getting an OSError read-only filesystem
- I am getting an OSError [Errno 24] Too many open files
- Notification emails have generic-host instead of host_name
- Thruk/Multisite reporting doesn't work using Shinken 1.2
- Pyro MemoryError during configuration distribution by the Arbiter to other daemons(satellites)

General Shinken troubleshooting steps to resolve common issue =

- Have you mixed installation methods! *Cleanup and install using a single method*.
- Have you installed the *check scripts and addon software*
- Is Shinken even running?
- Have you checked the *Shinken pre-requisites*?
- Have you *configured the WebUI module* in your shinken-specific.cfg file
- Have you *completed the Shinken basic configuration* and *Shinken WebUI configuration*
- Have you reviewed your Shinken centralized (*Simple-log broker module*) logs for errors
- Have you reviewed your Shinken daemon specific logs for errors or tracebacks (what the system was doing just before a crash)
- Have you reviewed your *configuration syntax* (keywords and values)
- Is what you are trying to use installed? Are its dependancies installed! Does it even work.
- Is what you are trying to use *a supported version*?
- Are you using the same Python Pyro module version on all your hosts running a Shinken daemon (You have to!)
- Are you using the same Python version on all your hosts running a Shinken daemon (You have to!)

- Have you installed Shinken with the SAME prefix (ex: /usr/local) on all your hosts running a Shinken daemon (You have to!)
- Have you enabled debugging logs on your daemon(s)
- How to identify the source of a Pyro MemoryError
- Problem with Livestatus, did it start, is it listening on the exppected TCP port, have you enabled and configured the module in shinken-specific.cfg.
- Have you installed the check scripts as the shinken user and not as root
- Have you executed/tested your command as the shinken user
- Have you manually generated check results
- Can you connect to your remote agent NRPE, NSClient++, etc.
- Have you defined a module on the wrong daemon (ex. NSCA receiver module on a Broker)
- Have you created a diagram illustrating your templates and inheritance
- System logs (/var/messages, windows event log)
- Application logs (MongoDB, SQLite, Apache, etc)
- Security logs (Filters, Firewalls operational logs)
- Use top or Microsoft Task manager or process monitor (Microsoft sysinternals tools) to look for memory, cpu and process issues.
- Use nagiostat to check latency and other core related metrics.
- Is your check command timeout too long
- Have you looked at your Graphite Carbon metrics
- Can you connect to the Graphite web interface
- Are there gaps in your data
- Have you configured your storage schema (retention interval and aggregation rules) for Graphite collected data.
- Are you sending data more often than what is expected by your storage schema.
- Storing data to the Graphite databases, are you using the correct IP, port and protocol, are both modules enabled; Graphite_UI and graphite export.

10.1.2 FAQ Answers

Review the daemon logs

A daemon is a Shinken process. Each daemon generates a log file by default. If you need to learn more about what is what, go back to *the shinken architecture*. The configuration of a daemon is set in the .ini configuration file(ex. brokerd.ini). Logging is enabled and set to level INFO by default.

Default log file location ‘’local_log=%(workdir)s/schedulerd.log’‘

The log file will contain information on the Shinken process and any problems the daemon encounters.

Changing the log level during runtime

shinken-admin is a command line script that can change the logging level of a running daemon.

‘’linux-server# ./shinken-admin ...’‘

Changing the log level in the configuration

Edit the <daemon-name>.ini file, where daemon name is pollerd, schedulerd, arbiterd, reactionnerd, receiverd. Set the log level to: DEBUG Possible values: DEBUG,INFO,WARNING,ERROR,CRITICAL

Re-start the Shinken process.

OSError read-only filesystem error

You poller daemon and reactionner daemons are not starting and you get a traceback for an OSError in your logs.

‘’OSError [30] read-only filesystem’‘

Execute a ‘mount’ and verify if /tmp or /tmpfs is set to ‘ro’ (Read-only). As root modify your /etc/fstab to set the filesystem to read-write.

OSError too many files open

The operating system cannot open anymore files and generates an error. Shinken opens a lot of files during runtime, this is normal. Increase the limits.

Google: changing the max number of open files linux / debian / centos / RHEL

cat /proc/sys/fs/file-max

su - shinken $ ulimit -Hn $ ulimit -Sn

This typically changing a system wide file limit and potentially user specific file limits. (ulimit, limits.conf, sysctl, sysctl.conf, cat /proc/sys/fs/file-max)

To immediately apply changes ulimit -n xxxxx now

Notification emails have generic-host instead of host_name

Try defining host_alias, which is often the field used by the notification methods.

Why does Shinken use both host_alias and host_name. Flexibility and historicaly as Nagios did it this way.

Reporting does not work with Shinken 1.2

Set your Scheduler log level to INFO by editing shinken/etc/scheduler.ini.

Upgrade to Shinken 1.2.1, which fixes a MongoDB pattern matching error.

How to identify the source of a Pyro MemoryError

Are the satellites identical in every respect? All the others work just fine? What is the memory usage of the scheduler after sending the configuration data for each scheduler? Do you use multiple realms? Does the memory use increase for each Scheduler?

Possible causes

1. Shinken Arbiter is not preparing the configuration correctly sending overlarge objects

2) there is a hardware problem that causes the error, for instance a faulty memory chip or bad harddrive sector. Run a hardware diagnostics check and a memtest (http://www.memtest.org/) on the failing device 3) a software package installed on the failing sattelite has become corrupted. Re-install all software related to Pyro, possibly the whole OS. 4) or perhaps, and probably very unlikely, that the network infrastructure (cables/router/etc) experience a fault and deliver corrupt packets to the failing sattelite, whereas the other sattelites get good data.. Do an direct server to server test or end to end test using iPerf to validate the bandwidth and packet loss on the communication path.

Other than that, here are some general thoughts. A MemoryError means: "Raised when an operation runs out of memory but the situation may still be rescued (by deleting some objects). The associated value is a string indicating what kind of (internal) operation ran out of memory. Note that because of the underlying memory management architecture (C"s malloc() function), the interpreter may not always be able to completely recover from this situation; it nevertheless raises an exception so that a stack traceback can be printed, in case a run-away program was the cause. "

5) Check on the server the actual memory usage of the Scheduler daemon. Another possible reason for malloc() to fail can also be memory fragmentation, which means that there's enough free RAM but just not a free chunk somewhere in between that is large enough to hold the required new allocation size. No idea if this could be the case in your situation, and I have no idea on how to debug for this.

It is not entirely clear to me where exactly the memoryerror occurs: is it indeed raised on the sattelite device, and received and logged on the server? Or is the server throwing it by itself?

6) Other avenues of investigation Try running the python interpreter with warnings on (-Wall). Try using the HMAC key feature of Pyro to validate the network packets. Try using Pyro's multiplex server instead of the threadpool server.

CHAPTER 11

Integration With Other Software

11.1 Integration Overview

11.1.1 Introduction

One of the reasons that Shinken is such a popular monitoring application is the fact that it can be easily integrated into your existing infrastructure. There are several methods of integrating Shinken with the management software you're already using and you can monitor almost any type of new or custom hardware, service, or application that you might have.

11.1.2 Integration Points

Important: This diagram is deprecated and illustrates legacy Nagios. Which has nothing to do with the new architecture. eck.

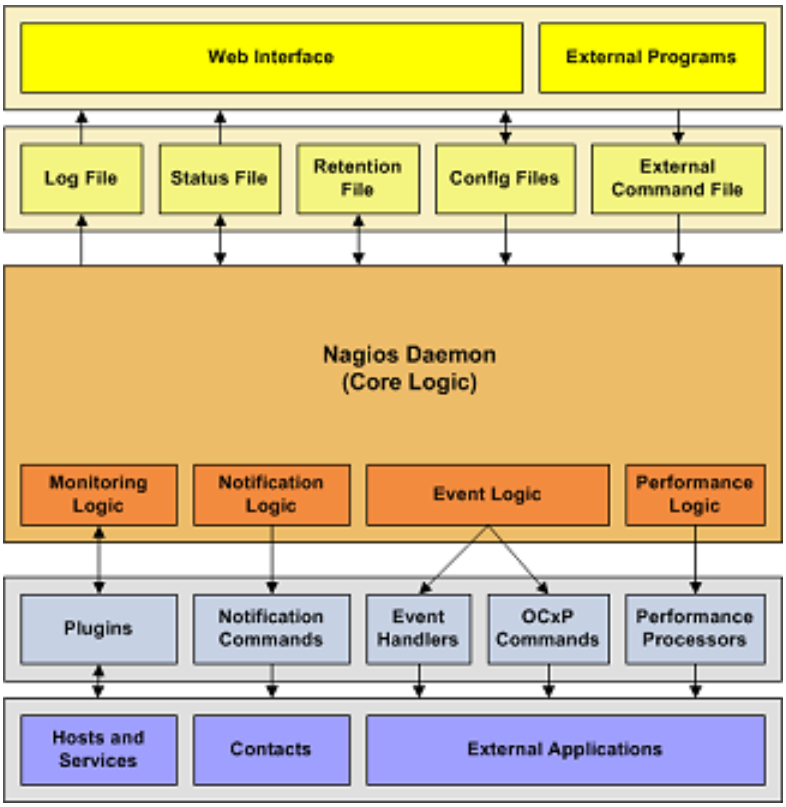

To monitor new hardware, services, or applications, check out the docs on:

- *Nagios Plugins*
- *Nagios Plugin API*
- *Passive Checks*
- *Event Handlers*

  It is also possible to use Shinken Poller daemon modules or Receiver daemon modules to provide daemonized high performance acquisition. Consult the Shinken architecture to learn more about poller modules. There are existing poller modules that can be usd as examples to further extend Shinken.

To get data into Nagios from external applications, check out the docs on:

- *Passive Checks*
- *External Commands*

To send status, performance, or notification information from Shinken to external applications, there are two typical paths. Through the Reactionner daemon which executes event handlers and modules or through the Broker daemon. The broker daemon provides access to all internal Shinken objects and state information. Thi data can be accessed through the Livestatus API. The data can also be forwarded by broker modules. Check out the docs on:

- *Broker modules*
- *Event Handlers*
- *OCSP* and *OCHP* Commands
- *Performance Data*
- *Notifications*

11.1.3 Integration Examples

I've documented some examples on how to integrate Shinken with external applications:

- *TCP Wrappers Integration* (security alerts)
- *SNMP Trap Integration* (Arcserve backup job status)

11.2 SNMP Trap Integration

11.2.1 Introduction

Nagios is not designed to be a replacement for a full-blown "SNMP" management application like HP OpenView or OpenNMS. However, you can set things up so that "SNMP" traps received by a host on your network can generate alerts in Nagios. Specific traps or groups of traps are associated with passive services.

As if designed to make the Gods of Hypocrisy die of laughter, "SNMP" is anything but simple. Translating "SNMP" traps and getting them into Nagios (as passive check results) can be a bit tedious. To make this task easier, I suggest you check out Alex Burger's "SNMP" Trap Translator project located at http://www.snmptt.org. When combined with Net-"SNMP", SNMPTT provides an enhanced trap handling system that can be integrated with Nagios.

You are strongly suggested to eventually have a logging system, SNMP manager or Hypervisor to classify and identify new alerts and events.

11.3 TCP Wrappers Integration

11.3.1 Introduction

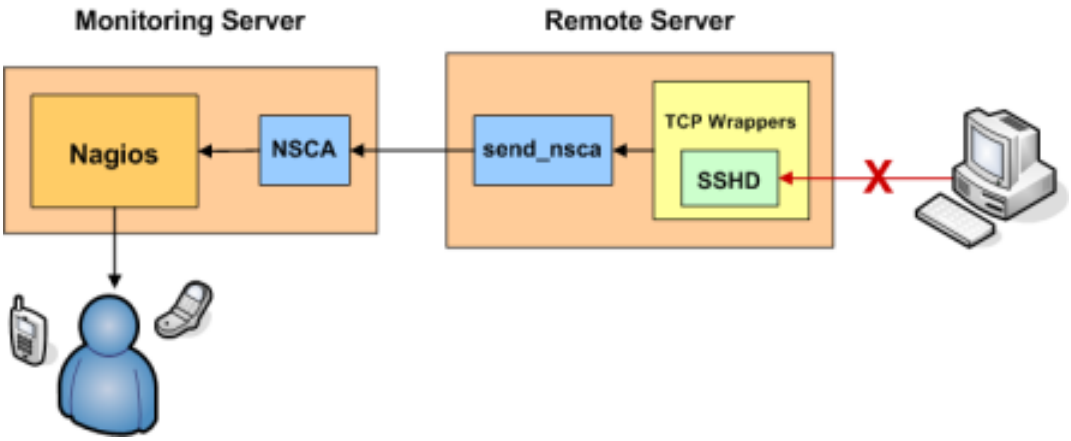

This document explains how to easily generate alerts in Shinken for connection attempts that are rejected by TCP wrappers. For example, if an unauthorized host attempts to connect to your "SSH" server, you can receive an alert in Shinken that contains the name of the host that was rejected. If you implement this on your Linux/Unix boxes, you'll be surprised how many port scans you can detect across your network.

These directions assume:

- You are already familiar with *Passive Checks* and how they work.
- You are already familiar with *Volatile Services* and how they work.
- The host which you are generating alerts for (i.e. the host you are using TCP wrappers on) is a remote host (called firestorm in this example). If you want to generate alerts on the same host that Shinken is running you will need to make a few modifications to the examples I provide.
- You have installed the NSCA daemon on your monitoring server and the NSCA client (**send_nsca**) on the remote machine that you are generating TCP wrapper alerts from.

11.3.2 Defining A Service

If you haven't done so already, create a *host definition* for the remote host (firestorm).

Next, define a service in one of your *object configuration files* for the TCP wrapper alerts on host firestorm. The service definition might look something like this:

```
define service{
    host_name firestorm
    service_description TCP Wrappers
    is_volatile 1
    active_checks_enabled 0
    passive_checks_enabled 1
    max_check_attempts 1
    check_command check_none
    ...
    }
```

There are some important things to note about the above service definition:

- The volatile option enabled. We want this option enabled because we want a notification to be generated for every alert that comes in.
- Active checks of the service as disabled, while passive checks are enabled. This means that the service will never be actively checked by Shinken - all alert information will have to be received passively from an external source.
- The "max_check_attempts" value is set to "1". This guarantees you will get a notification when the first alert is generated.

11.3.3 Configuring TCP Wrappers

Now you're going to have to modify the /etc/hosts.deny file on firestorm. In order to have the TCP wrappers send an alert to the monitoring host whenever a connection attempt is denied, you'll have to add a line similiar to the following:

```
ALL: ALL: RFC931: twist (/usr/local/shinken/libexec/eventhandlers/handle_tcp_wrapper %h %d)&
```

This line assumes that there is a script called handle_tcp_wrapper in the "/usr/local/shinken/libexec/eventhandlers/" directory on firestorm. We'll write that script next.

11.3.4 Writing The Script

The last thing you need to do is write the handle_tcp_wrapper script on firestorm that will send the alert back to the Shinken server. It might look something like this:

```
#!/bin/sh
/usr/local/shinken/libexec/eventhandlers/submit_check_result firestorm "TCP Wrappers" 2 "Denied $2-$1
```

Notice that the **handle_tcp_wrapper** script calls the **submit_check_result** script to actually send the alert back to the monitoring host. Assuming your Shinken server is called monitor, the **submit_check_result** script might look like this:

```
#!/bin/sh
# Arguments
#     $1 = name of host in service definition
#     $2 = name/description of service in service definition
#     $3 = return code
#     $4 = outputs

/bin/echo -e "$1\t$2\t$3\t$4\n" | /usr/local/shinken/bin/send_nsca monitor -c /usr/local/shinken/etc/
```

11.3.5 Finishing Up

You've now configured everything you need to, so all you have to do is restart the inetd process on firestorm and restart Shinken on your monitoring server. That's it! When the TCP wrappers on firestorm deny a connection attempt, you should be getting alerts in Shinken. The plugin output for the alert will look something like the following: "Denied sshd2-sdn-ar-002mnminnP321.dialsprint.net"

11.4 Thruk

11.4.1 Shinken installation

Create a new user shinken:

```
adduser shinken
```

Important: Be sure to create a home directory for shinken user. If not, you will not be able to start the shinken arbiter.

Next step retrieve the last version of shinken and uncompress it:

```
tar xfv shinken-0.5.1.tar.gz
```

Get into this directory and install it on your system:

```
sudo python setup.py install --install-scripts=/usr/bin
```

You will get new binaries into /usr/bin (files shinken-*) and some new directory (/etc/shinken, /var/lib/shinken).

Now, to unleash the daemons (ah ah ah!), you can use the script in init.d, or create your own script like:

```
root@Enclume:/etc/init.d# more shinken.sh
#!/bin/bash
cd /etc/init.d

for script in shinken-scheduler shinken-poller shinken-reactionner shinken-broker shinken-arbiter
do
  ./$script $1
done
```

Start your deamon:

```
/etc/init.d/shinken.sh start
```

Now check that the shinken processes are up and running:

```
patrice@Enclume:~/tmp/shinken-0.4$ ps -u shinken
PID TTY          TIME CMD
4358 ?        00:00:09 shinken-schedul
4367 ?        00:00:10 shinken-poller
4372 ?        00:00:00 shinken-poller
4380 ?        00:00:09 shinken-reactio
4385 ?        00:00:00 shinken-reactio
4949 ?        00:00:13 shinken-broker
4989 ?        00:00:00 shinken-poller
4990 ?        00:00:00 shinken-poller
4993 ?        00:00:00 shinken-poller
4996 ?        00:00:18 shinken-broker
4997 ?        00:00:00 shinken-broker
5001 ?        00:00:00 shinken-reactio
5004 ?        00:00:00 shinken-poller
5018 ?        00:00:10 shinken-arbiter
```

11.4.2 Configure Thruk

Get a fresh copy of Thruk (http://www.thruk.org/download.html) then uncompress your version and get into the root directory.

Now, create a new file named **thruk_local.conf**. Bellow, the content of this file:

```
~/tmp/Thruk-0.74$ cat thruk_local.conf
#####################################
```

```
# Backend Configuration, enter your backends here
<Component Thruk::Backend>
  <peer>
      name   = Shinken
      type   = livestatus
      <options>
          peer    = 127.0.0.1:50000
     </options>
  </peer>
#    <peer>
#        name   = External Icinga
#        type   = livestatus
#        <options>
#            peer    = 172.16.0.2:9999
#       </options>
#    </peer>
</Component>
```

Now launch the Thruk daemon:

```
~/tmp/Thruk-0.74/script$ ./thruk_server.pl
You can connect to your server at http://enclume:3000
```

Important: This article doesn't describe a true Thruk installation with Apache connection. Please refer Thruk documentation to get a cleaner installation.

Now, run your favorite internet browser with http://localhost:3000 and enjoy your Shinken installation !

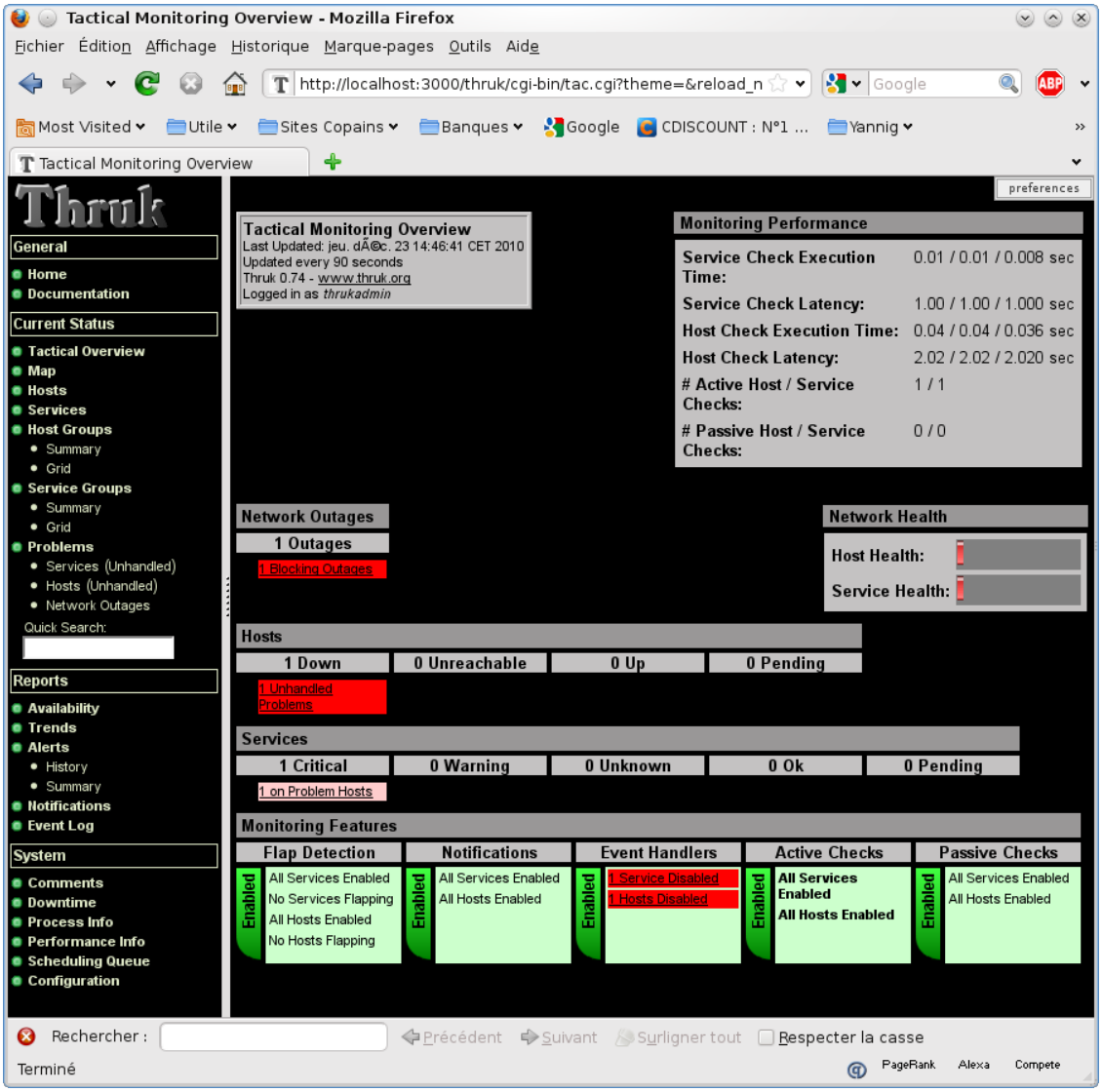

The parameters below are deprecated and are **only** useful if you use the old Nagios CGI UI.

11.5 Nagios CGI UI

11.5.1 Object Cache File

Format:

```
object_cache_file=<file_name>
```

Example:

```
object_cache_file=/usr/local/shinken/var/objects.cache
```

This directive is used to specify a file in which a cached copy of *Object Configuration Overview* should be stored. The cache file is (re)created every time Shinken is (re)started.

11.5.2 Temp File

Format:	temp_file=<file_name>
Example:	temp_file=/usr/local/nagios/var/nagios.tmp

This is a temporary file that Nagios periodically creates to use when updating comment data, status data, etc. The file is deleted when it is no longer needed.

11.5.3 Temp Path

Format:	temp_path=<dir_name>
Example:	temp_path=/tmp

This is a directory that Nagios can use as scratch space for creating temporary files used during the monitoring process. You should run **tmpwatch**, or a similiar utility, on this directory occassionally to delete files older than 24 hours.

11.5.4 Status File

Format:	status_file=<file_name>
Example:	status_file=/usr/local/nagios/var/status.dat

This is the file that Nagios uses to store the current status, comment, and downtime information. This file is used by the CGIs so that current monitoring status can be reported via a web interface. The CGIs must have read access to this file in order to function properly. This file is deleted every time Nagios stops and recreated when it starts.

11.5.5 Status File Update Interval

Format:	status_update_interval=<seconds>
Example:	status_update_interval=15

This setting determines how often (in seconds) that Nagios will update status data in the Status File. The minimum update interval is 1 second.

11.6 Thruk interface

Thruk is a web interface to get feedback about shinken status. You can also send command for asking shinken a new check, set a downtime on a host etc.

11.7 Use Shinken with ...

11.7.1 Shinken interface

Administrators need a means to view status data and interact with the system.

If you install Shinken using the *10 minutes installation* recommended method, you will have the **Shinken WebUI** installed. But it is not mandatory to use it, and you may prefer another interface. There are open-source and commercial web frontends using the **Livestatus API** or an **SQL backend** available to suit any needs.

11.7.2 Web interfaces

The choice of an interface starts with the method of data exchange: Those based on **Livestatus** and those based on an **SQL backend**.

The most responsive interfaces are the native **WebUI** and those based on **Livestatus**. The most scalable and flexible are those based on **Livestatus**.

SkonfUI is a discovery and configuration management UI that is **not production ready** and is meant as a beta preview for developers not users. Sorry.

Direct memory access based interface

- *Shinken WebUI*, included in the Shinken installation. Previews the Shinken features in an attractive package. Not meant for distributed deployments or scalability.

Livestatus based interfaces (Networked API)

- *Thruk*

Pickle based data export (Network)

- Complimentary module: *Graphite*
- *Note: Integrated out-of-the-box in :ref:'Shinken WebUI <use_with_webui>'*

Other

- Complimentary module: *PNP4Nagios* (Graphing interface)

Deprecated: Flat file export

- *Old CGI & VShell Note: The Nagios CGI web interface is deprecated.*

11.7.3 Which one is right for me?

Try them out and see which one fits best; this is especially easy with the Shinken WebUI and the Livestatus based interfaces.

- For users just starting out with small to medium installations, **Thruk** or **Shinken's WebUI** are good choices;
- For maximum scalability, intuitive UI and a solid feature set **Multisite** is recommended. **Thruk** is perl/PHP based UI that is very feature complete which also provides some scalability.

11.8 Use Shinken with Centreon

11.8.1 Centreon

Centreon is a famous French monitoring solution based on Nagios, which can also be used with Shinken.

11.8.2 How to use Shinken as a Centreon backend

The following Shinken Broker modules are required:

- NDO/MySQL
- Simple log
- Flat file perfdata

Below is the configuration you should set (there is already a sample configuration in your ''/etc/shinken/shinken-specific.cfg'' file)

Simple log

The module **simple_log** puts all Shinken's logs (Arbiter, Scheduler, Poller, etc.) into a single file.

In ''/etc/shinken/shinken-specific.cfg'':

```
define module{
```

```
module_name      Simple-log
module_type      simple_log
path             /var/lib/nagios/nagios.log
archive_path     /var/lib/nagios/archives/
```

}

It takes these parameters:

- module_name: name of the module called by the brokers
- module_type: simple_log
- path: path of the log file

NDO/MySQL

The module **ndodb_mysql** exports all data into a NDO MySQL database.

It needs the python module **MySQLdb** (Debian: ''sudo apt-get install python-mysqldb'', or ''easy_install MySQL-python'')

In ''/etc/shinken/shinken-specific.cfg'':

```
define module{
```

```
module_name      ToNdodb_Mysql
module_type      ndodb_mysql
database         ndo       ; database name
user             root      ; user of the database
password         root      ; must be changed
```

```
host             localhost ; host to connect to
character_set    utf8      ;optional, default: utf8
```

}

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Service Perfdata

The module **service_perfdata** exports service's perfdata to a flat file.

In ''/etc/shinken/shinken-specific.cfg'':

```
define module{
```

```
module_name      Service-Perfdata
module_type      service_perfdata
path             /var/lib/shinken/service-perfdata
```

}

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Configure Broker to use these modules

In ''/etc/shinken/shinken-specific.cfg" find the object **Broker**, and add the above modules to the **modules** line:

```
define broker{
```

```
     broker_name      broker-1
[...]
```

```
modules          Simple-log,ToNdodb_Mysql,Service-Perfdata
```

}

Configure Scheduler to match Centreon's Poller

Shinken's "Scheduler" is called a "Poller" in Centreon. If you keep the sample Scheduler name, you won't see any data in the Centreon interface.

So edit ''/etc/shinken/shinken-specific.cfg" and change the Scheduler name to match the Centreon's Poller name ("default"):

```
define scheduler{
```

```
scheduler_name   default
[...]
```

}

11.9 Use Shinken with Graphite

11.9.1 Graphite

- Homepage: Graphite
- Screenshots:
- presentation: http://pivotallabs.com/talks/139-metrics-metrics-everywhere
- Description: "Graphite is an easy to use scalable time-series database and a web API that can provide raw data for client rendering or server side rendering. It is the evolution of RRDtool."
- License: GPL v2
- Shinken dedicated forum: http://www.shinken-monitoring.org/forum/index.php/board,9.0.html
- Graphite dedicated forum: https://answers.launchpad.net/graphite

11.9.2 Install graphite

The best installation guide is actually this youtube walkthrough from Jason Dixon (Obfuscurity). There is a Chef recipe for the above demonstration.

See http://graphite.readthedocs.org/en/latest/install.html documentation tells what to install but not how to configure Apache and Django.

See Installing Graphite version 0.9.8 example. Just update for version 0.9.10 it is the exact same installation steps. The vhost example also worked for me, while the wsgi examples did not for me.

Make sure you set the timezone properly in ''/opt/graphite/webapp/graphite/local_settings.py'', for instance:

```
TIME_ZONE = 'Europe/Paris'</code>
```

11.9.3 Using Shinken with Graphite

The Shinken Broker module **graphite** is in charge of exporting performance data from Shinken to the Graphite databases.

Configure graphite module

<code>define module{

```
module_name     Graphite-Perfdata
module_type     graphite_perfdata
host            localhost
port            2003
templates_path  /usr/local/shinken/share/templates/graphite/
```

}

Additional list of options for the *Graphite export module and more in-depth documentation*.

Enable it

Edit ‘’/etc/shinken/shinken-specific.cfg” and find the **Broker** object, and add the graphite module to its **modules** line:

```
define broker{
```

```
     broker_name      broker-1
[...]
```

```
modules          Livestatus,Simple-log,WebUI,Graphite-Perfdata
```

}

Use it

With Shinken UI

Still in ‘’/etc/shinken/shinken-specific.cfg’‘, find the GRAPHITE_UI object and configure the URL to your Graphite install. If you used a graphite_data_source in the Graphite-Perfdata section, make sure to specify it here as well.

```
define module {
```

```
module_name GRAPHITE_UI
uri http://monitoring.mysite.com/graphite/
graphite_data_source shinken
...
```

}

Then find the WebUI object, and add GRAPHITE_UI to its modules (you’ll want to replace PNP_UI):

```
define module {
```

```
module_name WebUI
modules Apache_passwd, ActiveDir_UI, Cfg_password, GRAPHITE_UI, Mongodb
...
```

}

Restart Shinken to take the changes into account.

You have to possibility to use graphite template files. They are located in “templates_path”, (from the graphite_webui module) They are file containing graphite urls with shinken contextual variables. Ex :

‘’${uri}render/?width=586&height=308&target=alias(legendValue(${host}.${service}.’user’%2C%22last%22)%2C%22User%22)&ta

is used for check_cpu. Split this string using & as a separator to understand it. It's quite easy. Use graphite uri api doc.

Shinken uses the templates tht matches the check_command, like pnp does.

Important: The suggested configuration below is not final and has just been created, the documentation needs to be updated for the correct usage of the .graph templates used in WebUI. There are a number of the already created, see the existing packs to learn how to use them properly. Sorry for the inconvenience.

with Thruk

Thruk offers a proper integration with PNP, but not with Graphite. Still, you can use graphite with Thruk. Simply use the **action_url** for your service/host to link toward the graphite url you want. Use HOSTNAME and SERVICEDESC macros. The action_url icon will be a link to the graph in Thruk UI. For ex :

'' http://MYGRAPHITE/render/?lineMode=connected&width=586&height=308&_salt=1355923874.899&target=cactiStyle($HOSTNA

is what I use in my *Thruk*.

A change has been pushed in thruk's github to grant Thruk the features it has for pnp to graphite. The rule above (use action_url) still applies. Graphite will be displayed when the action_url contains the keyword "render".

Important: The graphite template files feature is not used in Thruk. It is a "shinken UI only" feature.

Enjoy it

Restart shinken-arbiter and you are done.

```
/etc/init.d/shinken-arbiter restart</code>
```

11.10 Use Shinken with Multisite

11.10.1 Check_MK Multisite

11.10.2 Using Shinken with Multisite

Multisite communicates with Shinken through the LiveStatus module. If you used the sample configuration, everything should be ready already. :)

You can review the configuration using the following steps.

Enable Livestatus module

See *enable Livestatus module*.

Configure Multisite

Latest versions of Multisite are included into Check_MK, which must be fully installed although we will only use the web interface.

To install and configure Multisite manually, follow **'instructions at MK website'_**.

Best choice is to use Shinken **:ref:'install script <shinken_10min_start#method_1the_easy_way>'** (In Shinken versions >1.0). With addons installation option (''./install -a multisite'') it is fast and easy to install and configure it as Multisite's default site.

> **Warning:** If you get some error installing Multisite related with unknown paths ("can not find Multisite_versionXXX") perhaps you must edit ''init.d/shinken.conf" file and adjust MKVER variable (search for "export MKVER") with current stable available version of Check_MK as stated on MK website.

Check_MK install quick guide

- Install check_mk: Detailed **'instructions'_** there. Shell driven install with a lot of questions related with Check_mk install paths and integration with Apache and existing "Nagios". For Shinken some default answers must be changed to accomodate Shinken install.
- Edit config file ''multisite.mk'', usually in ''/etc/check_mk'', to insert a new site pointing to Shinken and write Livestatus socket address as declared at *Shinken's Livestatus module*. Socket may also be an unix socket ("unix:/some/other/path").
- Restart Apache.

''/etc/check_mk/multisite.mk'':

```
sites = {
```

```
"munich": {
    "alias": "Munich"
},
"Shinken": {
   "alias":          "Shinken",
   "socket":         "tcp:127.0.0.1:50000",
   "url_prefix":     "http://shinken.fqdn/",
 },
```

} .. note:: Replace "shinken.fqdn" with the complete URI to reach Shinken host from browser (not 127.0.0.1!). Used by PNP4Nagios's mouse-over images.

If you plan to use Multisite only as web UI no more configuration is needed. Also you can disable WATO (Web Administration TOol) by including the line **wato_enabled = False** in ''multisite.mk''. .. _instructions at MK website: http://mathias-kettner.de/checkmk_multisite_setup.html .. _instructions: http://mathias-kettner.de/checkmk_getting_started.html

11.11 Use Shinken with Nagvis

11.11.1 NagVis

11.11.2 Using Shinken with NagVis

NagVis communicates with Shinken through the LiveStatus module. If you used the sample configuration, everything should be ready already. :)

You can review the configuration using the following steps.

Enable Livestatus module

The Livestatus API is server from the Shinken broker. It permits communications via TCP to efficiently retrieve the current state and performance data of supervised hosts and services from Shinken. It also exposes configuration information.

See *enable Livestatus module*.

Nagvis Installation

Download the software and follow the installation guide from http://www.nagvis.org/

NagVis configuration

Nagvis needs to know where the Shinken Livestatus API is hosted.

In NagVis configuration file ‘’/etc/nagvis/nagvis.ini.php’‘:

```
[backend_live_1]
backendtype="mklivestatus"
htmlcgi="/nagios/cgi-bin"
```

socket=”tcp:localhost:50000“

Important: If you are using a non local broker (or a distributed Shinken architecture with multiple brokers) you should change **localhost** to the **IP/Servername/FQDN of your broker**!

11.12 Use Shinken with Old CGI and VShell

11.12.1 For the Old CGI & VShell

The Old CGI and VShell uses the old flat file export. Shinken can export to this file, but beware: this method is very very slooooow!

Warning: You should migrate to a Livestatus enabled web interface.

11.12.2 Declare the status_dat module

Export all status into a flat file in the old Nagios format. It’s for small/medium environment because it’s very slow to parse. It can be used by the Nagios CGI. It also exports the objects.cache file for this interface.

Edit your /etc/shinken/shinken-specific.cfg file:

```
define module{
```

```
     module_name              Status-Dat
     module_type              status_dat
     status_file              /var/lib/shinken/status.data
     object_cache_file        /var/lib/shinken/objects.cache
     status_update_interval   15 ; update status.dat every 15s
}
```

11.12.3 Enable it

Edit your /etc/shinken/shinken-specific.cfg file and find the object Broker:

```
define broker{
    broker_name      broker-1
[...]
    modules          Simple-log,Status-Dat
}
```

11.13 Use Shinken with PNP4Nagios

11.13.1 PNP4Nagios

11.13.2 Install PNP4Nagios automatically

You can use the Shinken install script to install everything automatically (if your distro is supported):

```
./install -a pnp4nagios
```

By default PNP4Nagios is installed in ''/usr/local/pnp4nagios''. If you prefer another location, edit ''PNPPREFIX" in ''install.d/shinken.conf''.

11.13.3 Install PNP4Nagios manually

See PHP4Nagios installation documentation.

In a nutshell:

```
./configure --with-nagios-user=shinken --with-nagios-group=shinken
make all
```

make fullinstall

Don't forget to make PNP4Nagios' npcd daemon to start at boot, and launch it:

```
chkconfig --add npcd # On RedHat-like
update-rc.d npcd defaults # On Debian-like
```

/etc/init.d/npcd start

Configure npcdmod

The module **npcdmod** is in charge to export performance data from Shinken to PNP.

```
define module{
```

```
module_name       NPCDMOD
module_type       npcdmod
config_file       <PATH_TO_NPCD.CFG>
```

}

Don't forget to replace "<PATH_TO_NPCD.CFG>" with your own value; By default something like ''/usr/local/pnp4nagios/etc/npcd.cfg''.

Enable it

Edit ‘’/etc/shinken/shinken-specific.cfg” and find the object **Broker** to add above defined “NPCDMOD” to its **modules** line:

```
define broker{
```

```
     broker_name      broker-1
[...]
```

```
modules          Simple-log,NPCDMOD
```

}

Edit ‘’/etc/shinken/shinken-specific.cfg” and find the object **WebUI** to add above defined “PNP_UI” to its **modules** line:

```
define broker{
```

```
     module_name      WebUI
[...]
```

```
modules          Apache_passwd,ActiveDir_UI,Cfg_password,PNP_UI
```

}

Then restart broker :

```
# /etc/init.d/shinken-broker restart</code>
```

11.13.4 Share users with Thruk

Edit ‘’/etc/httpd/conf.d/pnp4nagios.conf” (RedHat path) and replace AuthName and AuthUserFile with: <code> AuthName “Thruk Monitoring” AuthUserFile /etc/thruk/htpasswd

Then restart Apache:

```
service httpd restart
```

11.13.5 Set the action_url option

In order to get the graphs displayed in *Thruk*, you need to set the **action_url** option in *host* and *service* definitions, and it must include the string “/pnp4nagios/” (Thruk doc).

If you want the link and the graph for all hosts and services, you could set the option directly in the default templates, in ‘’templates.cfg’‘:

```
define host{
```

```
      name                            generic-host
[...]
```

```
      process_perf_data               1
[...]
```

```
      #action_url                     http://<PNP4NAGIOS_HOST>/pnp4nagios/graph?host=$HOSTNAME$
      # If not an absolute URI, it must be relative to /cgi-bin/thruk/, not /thruk/!
      action_url                      ../../pnp4nagios/graph?host=$HOSTNAME$
[...]
define service{
```

```
      name                            generic-service
[...]
```

```
      process_perf_data               1
[...]
```

```
#action_url                      http://<PNP4NAGIOS_HOST>/pnp4nagios/graph?host=$HOSTNAME$&srv=$SERVI
# If not an absolute URI, it must be relative to /cgi-bin/thruk/, not /thruk/!
action_url                      ../../pnp4nagios/graph?host=$HOSTNAME$&srv=$SERVICEDESC$
```

Don't forget to replace "<PNP4NAGIOS_HOST>" with the server IP/name running PNP4Nagios (Don't replace $HOSTNAME$ and $SERVICEDESC$!)

Make sure to also have **process_perf_data** set to **1** for both hosts and services.

11.13.6 Link back to Thruk

Ask PNP4Nagios to link to ''/thruk/cgi-bin" rather than ''/nagios/cgi-bin'':

```
sed -i -e 's,/nagios/cgi-bin,/thruk/cgi-bin,' /opt/pnp4nagios/etc/config_local.php
```

11.13.7 Enjoy it

Restart shinken-arbiter and you are done.

```
/etc/init.d/shinken-arbiter restart</code>
```

11.14 Use Shinken with Thruk

11.14.1 Thruk

11.14.2 Install Thruk

See Thruk installation documentation.

Note: if you're using SELinux, also run:

```
chcon -t httpd_sys_script_exec_t /usr/share/thruk/fcgid_env.sh
chcon -t httpd_sys_script_exec_t /usr/share/thruk/script/thruk_fastcgi.pl
chcon -R -t httpd_sys_content_rw_t /var/lib/thruk/
chcon -R -t httpd_sys_content_rw_t /var/cache/thruk/
chcon -R -t httpd_log_t /var/log/thruk/
setsebool -P httpd_can_network_connect on
```

11.14.3 Using Shinken with Thruk

Thruk communicates with Shinken through the LiveStatus module. If you used the sample configuration, everything should be ready already. :)

You can review the configuration using the two following steps.

Enable Livestatus module

See *enable Livestatus module*.

Declare Shinken peer in Thruk

Edit ''/etc/thruk/thruk_local.conf" and declare the Shinken peer:

```
enable_shinken_features = 1
<Component Thruk::Backend>
    <peer>
        name   = External Shinken
        type   = livestatus
        <options>
            peer    = 127.0.0.01:50000
        </options>
        # Uncomment the following lines if you want to configure shinken through Thruk
        #<configtool>
        #    core_type      = shinken
        #    core_conf      = /etc/shinken/shinken.cfg
        #    obj_check_cmd  = service shinken check
        #    obj_reload_cmd = service shinken restart
        #</configtool>
    </peer>
</Component>
```

Or use the backend wizard which starts automatically upon first installation.

Don't forget to change the 127.0.0.1 with the IP/name of your broker if it is installed on an different host, or if you are using a distributed architecture with multiple brokers!

Credit Shinken in the webpages title :-)

Edit ''/etc/thruk/thruk.conf'':

```
title_prefix = Shinken+Thruk-
```

Configure users

Passwords are stored in ''/etc/thruk/htpasswd" and may be modified using the ''htpasswd" command from Apache:

```
htpasswd /etc/thruk/htpasswd your_login
```

User permissions: modify ''templates.cfg:generic-contact'':

```
# I couldn't manage to get Thruk-level permissions to work, let's use Shinken admins privileges
can_submit_commands             0
```

and define some users as admins in the Shinken configuration:

```
define contact {

  # ...

  use             generic-contact
  is_admin        1

  # ...
}
```

Allow Thruk to modify its configuration file:

```
chgrp apache /etc/thruk/cgi.cfg
chmod g+w /etc/thruk/cgi.cfg
```

Set permissions for your users in Config Tool > User Settings > **authorized_for_**...

11.14.4 Using PNP4Nagios with Thruk

See *PNP4Nagios*.

11.15 Use Shinken with WebUI

11.15.1 Shinken WebUI

Shinken includes a self sufficient Web User Interface, which includes its own web server (No need to setup Apache or Microsoft IIS)

Shinken WebUI is started at the same time Shinken itself does, and is configured using the main Shinken configuration file by setting a few basic parameters.

11.15.2 Set up the WebUI module

Enable the **webui** module in ''shinken-specific.cfg" configuration file that is on the server where your **Arbiter** is installed.

```
define module{
```

```
module_name      WebUI
module_type      webui

host             0.0.0.0       ; mean all interfaces of your broker server
port             7767

# CHANGE THIS VALUE or someone may forge your cookies
auth_secret      TOCHANGE

# Allow or not the html characters in plugins output
# WARNING: so far, it can be a security issue
allow_html_output  0

# Option welcome message
```

```
#login_text             Welcome to ACME Shinken WebUI.

#http_backend    auto
# ; can be also: wsgiref, cherrypy, paste, tornado, twisted
# ; or gevent. auto means best match in the system.
modules          Apache_passwd,ActiveDir_UI,Cfg_password,Mongodb

# Modules available for the WebUI:
#
#   Note: Choose one or more authentication methods.
#
#   Apache_passwd: use an Apache htpasswd files for auth
#   ActiveDir_UI: use AD for auth and photo collect
#   Cfg_password: use passwords in contacts configuration for authentication
#
#   PNP_UI: Use PNP graphs in the UI
#   GRAPHITE_UI: Use graphs from Graphite
#
#   Mongodb: Necessary for enabling user preferences in WebUI
```

}

Important: Have you changed the **auth_secret** parameter already? No? Do it now!

Note: The web-server handling HTTP Request to the WebUI is a Python process. You *do not need* any web-server (like Apache) to run the WebUI.

11.15.3 Authentification modules

The WebUI use modules to lookup your user password and allow to authenticate or not.

By default it is using the **cfg_password_webui** module, which will look into your contact definition for the **password** parameter.

Tip: You need to declare these modules in the **modules** property of WebUI.

Shinken contact - cfg_password_webui

The simpliest is to use the users added as Shinken contacts.

```
define module{
```

```
     module_name Cfg_password
     module_type cfg_password_webui
}
```

Apache htpasswd - passwd_webui

This module uses an Apache passwd file (htpasswd) as authentification backend. All it needs is the full path of the file (from a legacy Nagios CGI installation, for example).

```
define module{
```

```
     module_name      Apache_passwd
     module_type      passwd_webui

     # WARNING: put the full PATH for this value!
     passwd           /etc/shinken/htpasswd.users
}
```

Check the owner (must be Shinken user) and mode (must be readable) of this file.

If you don't have such a file you can generate one with the "htpasswd" command (in Debian's "apache2-utils" package), or from websites like htaccessTools.

Important: To be able to log into the WebUI, users also have to be Shinken contacts! So adding an user in this file without adding it in the contacts will have no effect.

Active Directory / OpenLDAP - ad_webui

This module allows to lookup passwords into both Active Directory or OpenLDAP entries.

```
define module {
```

```
  module_name ActiveDir_UI
  module_type ad_webui
  ldap_uri ldaps://adserver
  username user
  password password
  basedn DC=google,DC=com

  # For mode you can switch between ad (active dir)
  # and openldap
  mode        ad
}
```

Change "adserver" by your own dc server, and set the "user/password" to an account with read access on the basedn for searching the user entries.

Change "mode" from "ad" to "openldap" to make the module ready to authenticate against an OpenLDAP directory service.

You could also find module sample in shinken.specific.cfg.

User photos

In the WebUI users can see each others photos.

At this point only the "ad_webui" module allows to import and display photos in the WebUI. There is no configuration: if you add the "ad_webui" module it will import contact photos automatically.

11.15.4 User preferences modules

The WebUI use mongodb to store all user preferences, dashboards and other information.

To enable user preferences do the following:

- install mongodb using the Shinken installation script: cd /usr/local/shinken ; ./install -a mongodb
- add "Mongodb" to your WebUI module list as done in the example at the top of this page

11.15.5 Metrology graph modules

You can link the WebUI so it will present graphs from other tools, like *PNP4Nagios* or Graphite. All you need is to declare such modules (there are already samples in the default configuration) and add them in the WebUI **modules** definition.

PNP graphs

You can ask for a PNP integration with a **pnp_webui** module. Here is its definition:

```
# Use PNP graphs in the WebUI
define module{
```

```
module_name    PNP_UI
module_type    pnp_webui
uri            http://YOURSERVERNAME/pnp4nagios/       ; put the real PNP uri here. YOURSERVERNAME mu
                                                 ; to the hostname of the PNP server
}
```

Shinken will automatically replace YOURSERVERNAME with the broker hostname at runtime to try and make it work for you, but you MUST change it to the appropriate value.

Graphite graphs

You can ask for Graphite graphs with the **graphite_ui** definition.

```
define module{
```

```
module_name    GRAPHITE_UI
module_type    graphite_webui
uri            http://YOURSERVERNAME/ ; put the real GRAPHITE uri here. YOURSERVERNAME must be change
                                      ; to the hostname of the GRAPHITE server
}
```

Shinken will automatically replace YOURSERVERNAME with the broker hostname at runtime to try and make it work for you, but you MUST change it to the appropriate value.

11.15.6 Use it!

The next step is very easy: just access the WebUI URI (something like %%http://127.0.0.1:7767/%%) on log in with the user/password set during the previous part! The default username and password is admin/admin

CHAPTER 12

Security and Performance Tuning

12.1 Security Considerations

12.1.1 Introduction

Your monitoring box should be viewed as a backdoor into your other systems. In many cases, the Nagios server might be allowed access through firewalls in order to monitor remote servers. In most all cases, it is allowed to query those remote servers for various information. Monitoring servers are always given a certain level of trust in order to query remote systems. This presents a potential attacker with an attractive backdoor to your systems. An attacker might have an easier time getting into your other systems if they compromise the monitoring server first. This is particularly true if you are making use of shared "SSH" keys in order to monitor remote systems.

If an intruder has the ability to submit check results or external commands to the Nagios daemon, they have the potential to submit bogus monitoring data, drive you nuts you with bogus notifications, or cause event handler scripts to be triggered. If you have event handler scripts that restart services, cycle power, etc. this could be particularly problematic.

Another area of concern is the ability for intruders to sniff monitoring data (status information) as it comes across the wire. If communication channels are not encrypted, attackers can gain valuable information by watching your monitoring information. Take as an example the following situation: An attacker captures monitoring data on the wire over a period of time and analyzes the typical CPU and disk load usage of your systems, along with the number of users that are typically logged into them. The attacker is then able to determine the best time to compromise a system and use its resources (CPU, etc.) without being noticed.

Here are some tips to help ensure that you keep your systems secure when implementing a Nagios-based monitoring solution...

12.1.2 Best Practices

- **Use a Dedicated Monitoring Box**. I would recommend that you install Nagios on a server that is dedicated to monitoring (and possibly other admin tasks). Protect your monitoring server as if it were one of the most important servers on your network. Keep running services to a minimum and lock down access to it via TCP wrappers, firewalls, etc. Since the Nagios server is allowed to talk to your servers and may be able to poke through your firewalls, allowing users access to your monitoring server can be a security risk. Remember, its always easier to gain root access through a system security hole if you have a local account on a box.

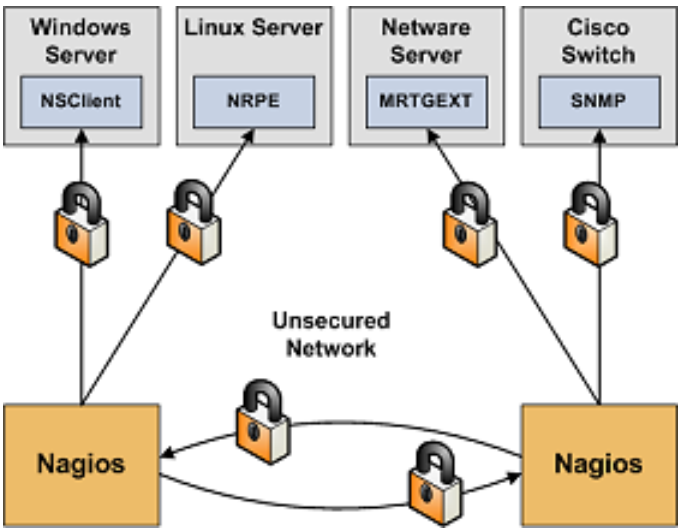

12.2 Tuning Shinken For Maximum Performance

12.2.1 Introduction

So you've finally got Shinken up and running and you want to know how you can tweak it a bit. Tuning Shinken to increase performance can be necessary when you start monitoring a large number (> 10,000) of hosts and services. Here are the common optimization paths.

12.2.2 Designing your installation for scalability

Planning a large scale Shinken deployments starts before installing Shinken and monitoring agents.

Scaling Shinken for large deployments

12.2.3 Optimization Tips:

- Graph your shinken server performance. In order to keep track of how well your installation handles load over time and how your configuration changes affect it, you should be graphing several important statistics. This is really, really, really useful when it comes to tuning the performance of an installation. Really. Information on how to do this can be found :ref:'Graphing Performance Info With MRTG <securityandperformancetuning-mrtggraphs>'here.
- Check service latencies to determine best value for maximum concurrent checks. Nagios can restrict the number of maximum concurrently executing service checks to the value you specify with the max_concurrent_checks option. This is good because it gives you some control over how much load Nagios will impose on your monitoring host, but it can also slow things down. If you are seeing high latency values (> 10 or 15 seconds) for the majority of your service checks. That's not Shinken's fault - its yours. Under ideal conditions, all service checks would have a latency of 0, meaning they were executed at the exact time that they were scheduled to be executed. However, it is normal for some checks to have small latency values. I would recommend taking the minimum number of maximum concurrent checks reported when running Shinken with the -s command line argument and doubling it. Keep increasing it until the average check latency for your services is fairly low. More information on service check scheduling can be found :ref:'Service and Host Check Scheduling <advancedtopics-checkscheduling>'here.
- Use passive checks when possible. The overhead needed to process the results of :ref:'Passive Checks <thebasics-passivechecks>'passive service checks is much lower than that of "normal" active checks, so make use of that piece of info if you're monitoring a slew of services. It should be noted that passive service checks are only really useful if you have some external application doing some type of monitoring or reporting, so if you're having Nagios do all the work, this won't help things.
- Avoid using interpreted plugins. One thing that will significantly reduce the load on your monitoring host is the use of compiled (C/C++, etc.) plugins rather than interpreted script (Perl, etc) plugins. While Perl scripts and such are easy to write and work well, the fact that they are compiled/interpreted at every execution instance can significantly increase the load on your monitoring host if you have a lot of service checks. If you want to use Perl plugins, consider compiling them into true executables using perlcc(1) (a utility which is part of the standard Perl distribution).
- Optimize host check commands. If you're checking host states using the **check_ping** plugin you'll find that host checks will be performed much faster if you break up the checks. Instead of specifying a max_attempts value of 1 in the host definition and having the **check_ping** plugin send 10 "ICMP" packets to the host, it would be much faster to set the max_attempts value to 10 and only send out 1 "ICMP" packet each time. This is due to the fact that Nagios can often determine the status of a host after executing the plugin once, so you want to make the first check as fast as possible. This method does have its pitfalls in some situations (i.e. hosts that are slow to respond may be assumed to be down), but you'll see faster host checks if you use it. Another option would be to use a

faster plugin (i.e. **check_fping**) as the host_check_command instead of **check_ping**. - Don't use agressive host checking. Unless you're having problems with Shinken recognizing host recoveries, I would recommend not enabling the use_aggressive_host_checking option. With this option turned off host checks will execute much faster, resulting in speedier processing of service check results. However, host recoveries can be missed under certain circumstances when this it turned off. For example, if a host recovers and all of the services associated with that host stay in non-OK states (and don't "wobble" between different non-OK states), Shinken may miss the fact that the host has recovered. A few people may need to enable this option, but the majority don't and I would recommendnot using it unless you find it necessary.

- Optimize hardware for maximum performance. Hardware performance shouldn't be an issue unless: - you're monitoring thousands of services - you are writing to a metric database such as RRDtool or Graphite. Disk access will be a very important factor. - you're doing a lot of post-processing of performance data, etc. Your system configuration and your hardware setup are going to directly affect how your operating system performs, so they'll affect how Shinken performs. The most common hardware optimization you can make is with your hard drives, RAID, do not update attributes for access-time/write-time. - Shinken needs quite a bit of memory which is pre-allocated by the Python processes. - Move your Graphite metric databases to dedicated servers. Use multiple carbon-relay and carbon-cache daemons to split the load on a single server.

12.3 Scaling a Shinken installation

12.3.1 Introduction

Shinken is designed to scale horizontally, but carefully planning your deployment will improve chances of success.

12.3.2 Scalability guide

Learn how to prepare by reading the main scalability guide for large Shinken installations

12.4 Shinken performance statistics

12.4.1 Introduction

Shinken provides some statistics in the log files on the health of the Shinken services. These are not currently available in the check_shinken check script. Support is planned in a future release. This will permit graphical review that Shinken :

- Operates efficiently
- Locate problem areas in the monitoring process
- Observe the performance impacts of changes in your Shinken configuration

Shinken internal metrics are collected in the poller log and scheduler logs when *debug log level is enabled*.

CHAPTER 13

How to monitor ...

13.1 Monitoring an Asterisk server

Abstract

This document describes how you can monitor an Asterisk server. This monitoring covers typically:

- overall state

13.1.1 Introduction

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that was installed if you followed the quickstart.

13.1.2 Overview

Monitoring an asterisk server is possible with the check_sip plugin and an account on the Asterisk server.

13.1.3 Prerequisites

Have a valid account on the Asterisk device.

13.1.4 Steps

There are several steps you'll need to follow:

- Setup the check_sip plugin
- Add the good asterisk template to your host in the configuration
- Restart the Shinken Arbiter

13.1.5 What's Already Been Done For You

To make your life a bit easier, configuration templates are provided as a starting point:

- A selection of **check_sip** based commands definitions have been added to the "commands.cfg" file.
- An Asterisk host template is included the "templates.cfg" file. This allows you to add new host definitions in a simple manner.

The above-mentioned config files can be found in the ///etc/shinken/packs/network/services/asterisk// directory. You can modify the definitions in these and other templates to suit your needs. However, wait until you're more familiar with Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your devices in no time.

13.1.6 Setup the check_sip plugin

As the shinken account in your shinken server run:

```
wget "http://www.bashton.com/downloads/nagios-check_sip-1.2.tar.gz"
tar xvfz nagios-check_sip-1.2.tar.gz
cd nagios-check_sip-1.2/
cp check_sip /var/lib/nagios/plugins/
chmod a+x /var/lib/nagios/plugins/check_sip
```

13.1.7 Add tje SIP user credentials

In the file ///etc/shinken/packs/network/services/asterisk//macros you can edit the SIPUSER that you want ot use for the connection.

13.1.8 Declare your host in Shinken

The Asterisk template name is *asterisk*. All you need is to add it on yourhost.

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Windows machine.

We will suppose here that your server is named *srv-sip-1*. Of course change this name with the real name of your server.

Find your host definition and edit it:

Under Linux:

```
::
```

linux:~ # vi /etc/shinken/hosts/srv-sip-1.cfg

Or Windows:

```
::
```

c:wordpad c:shinkenetchostssrv-sip-1.cfg

All you need it to add the asterisk template for your host.

```
::
```

define host{ use asterisk,windows host_name srv-sip-1 address srv-sip-1.mydomain.com }

- The use sip is the "template" line. It mean that this host will **inherits** properties from this template. * the host_name is the object name of your host. It must be **unique**. * the address is ... the network address of your host :)

What is checked with an askterisk template?

At this point, you configure your host to be checked with an asterisk templates. What does it means? It means that you got some

- overall state of the asterisk server

13.1.9 Restarting Shinken

You're done with modifying the Shinken configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.2 check_wmi_plus.pl for shinken on windows

The premier WMI check program, check_wmi_plus.pl, can now be used on the windows platform in association with the shinken WMIC.EXE.

WMIC.EXE is not the native wmic of the windows system. It is a standalone program created to have the same input/output as the linux WMIC. This permits check_wmi_plus to use WMIC on windows with the same options as the linux program. WMIC.EXE (binaries and sources) are included with Shinken (1.2.2 and newer).

WMIC.EXE and associated Shinken programs under Windows, use the .NET framework 2.0.

13.2.1 Pre-requisites for using check_wmi_plus.pl

Check_wmi_plus.pl needs a perl interpreter to work, we recommend activePerl. At this time, strawberry perl cannot be used as a non interactive program. The Shinken poller starts programs in a non interactive mode, which means no Strawberrt perl.

- Download activeperl and select the 5.16.x Windows x64 version
- Download and install PERL Number::Format using CPAN
- Download and install PERL Config::Inifiles using CPAN
- Download and install PERL DataTime using CPAN
- Download and install .NET 2.0 framework from Microsoft (If it is not already installed). On Windows 2008 R2, install 3.5 (SP1) Framework. It includes the 2.0 version.

Important: PERL Dependencies for check_wmi_plus.pl (needed by check_wmi_plus and not installed by default. Please lookup the CPAN documentation if you don"t know how to install PERL modules)

After having installed all dependencies, you can now proceed to install check_wmi_plus.pl

- Download and install check_wmi_plus.pl

Make sure to change the references into the check_wmi_plus.conf file and the initial variables into the check_wmi_plus.pl (take a look to match the install folder and the WMIC.exe path)

13.2.2 Shinken configuration to use check_wmi_plus

At first you must configure monitoring parameters in the Shinken etc/resource.cfg file :

$DOMAIN$=domain

$DOMAINUSERSHORT$=shinken_user

$DOMAINUSER$=$DOMAIN$\$DOMAINUSERSHORT$

$DOMAINPASSWORD$=superpassword

$LDAPBASE$=dc=eu,dc=society,dc=com

These options are set by default, but the poller will also use a « hidden » *$DOMAINUSERSHORT$* parameter set in the Shinken templates.cfg. Just set this parameter in the resource.cfg file to overload the template and make use of a single configuration file.

These options are used by the check_wmi_plus plugin as credentials for WMI local or remote access. If you intend to monitor hosts that are not controlled by a domain controller (simple workgroup members) it is necessary to overload the *$DOMAINUSER$* parameter :

$DOMAIN$=domain

$DOMAINUSERSHORT$=shinken_user

$DOMAINUSER$=$DOMAINUSERSHORT$

$DOMAINPASSWORD$=superpassword

$LDAPBASE$=dc=eu,dc=society,dc=com

You need to set the appropriate values for your environment, the above values (domain, shinken_user and superpassword) are simply examples. :-)

To test the settings, just add a host to be monitored by setting a new host in a cfg file named with the name of the host for example etchostsclishinken.cfg based on the windows template:

define host{ use windows contact_groups admins host_name clishinken address clishinken icon_set server }

Restart the Shinken windows services. The WebUI now checks a new windows host : clishinken ! In this configuration, the remote WMI is executed using the credentials set into the resource.cfg file. It"s working but is not really secure.

> **Warning:** warning
> If you look in the local configuration you will see check_wmi_plus. For Windows, it's not allowed to use credentials for a local WMI request. As a workaround, WMIC.EXE will use the local credentials if the target host is set to localhost and the user is set to local. Check the etccommands-windows.cfg to see how it works. You will see something like this :
> define command {

```
 command_name    check_local_disks
    command_line    $PLUGINSDIR$/check_wmi_plus.pl -H localhost -u "local" -p "local" -m checkdrives
}
```

13.2.3 Secure method to use check_wmi_plus on windows

There is a new option to use check_wmi_plus with shinken : Use the poller configuration service credentials.

For most sysadmins, putting an unencrypted password in a config file is not a best practice. IT secuirty will be compromised if someone can read the configuration file. The rights of this user on servers are also very high (WMI requests need more configuration to be security compliant on windows server, including DCOM configuration or admin rights. . .) You can find a good idea of how to configure the remote wmi on windows here:

www.op5.com/how-to/agentless-monitoring-windows-wmi/

To set a "bypass" credential, only set the $DOMAINUSERSHORT$ to #### (4 sharp symbols without spaces) If the WMIC see this specific user,it just will use the caller credentials - in all cases the poller service user. By default, the Shinken services are set to localsystem.

Stop the services (manually or using the binstop_allservices.cmd script).

Open the services.msc (or the server manager, and then the services part)

double-click on the Shinken poller service

go to the log On tab

check the "This account" radio button and set the Shinken user account (the same as you set the resource.cfg file)

As you can see, you never see the password... Click on the Apply button (the first time you set an account to logon as a service, you will see a message box to announce the fact that the account is granted to logon as a service). Change the resource.cfg file to set the #### as the domainusershort and put a wrong password to be sure to remove the old credentials. Save the resource.cfg file.

Restart the services (manually or using the binstart_allservices.cmd)

The poller will now launch the WMI request under its own service account...

Important: Setting the remote WMI configuration on windows is not as easy as it seems.

The domains admins or other IT admins may set GPO or other tools to change the configuration of the system - including the right to enable or disable remote WMI. Please be patient, and change options one by one if your wmi tests are not working.

13.3 Monitoring Active Directory

Abstract

This document describes how you can monitor domain controller. This monitoring covers typically:

- Domain replication
- etc...

13.3.1 Introduction

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that was installed if you followed the quickstart.

13.3.2 Overview

Monitoring a domain controller is possible in an agentless way by polling via the network using the WMI protocol, like we proposed in the windows template.

13.3.3 Prerequisites

Have a valid account on the Microsoft Windows device (local or domain account) you will monitor using WMI queries and already get the windows server monitor with the windows template.

13.3.4 Steps

There are several steps you'll need to follow in order to monitor a Microsoft Exchange server.

- Add the good domain controller template to your windows host in the configuration

- Restart the Shinken Arbiter

13.3.5 What's Already Been Done For You

To make your life a bit easier, configuration templates are provided as a starting point:

- A selection of **check_ad_replication*** based commands definitions have been added to the "commands.cfg" file. This allows you to use the **check_wmi_plus** plugin.
- Some Exchange host templates are included the "templates.cfg" file. This allows you to add new host definitions in a simple manner.

The above-mentioned config files can be found in the ///etc/shinken/packs/microsoft/dc// directory. You can modify the definitions in these and other templates to suit your needs. However, wait until you're more familiar with Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your devices in no time.

13.3.6 Setup the check_wmi_plus plugin

If you already follow the *windows monitoring* tutorial, you should already got the check_wmi_plus plugin. If it's not done, please do it first.

13.3.7 Declare your host in Shinken

The domain controller template name is *dc*. All you need is to add it on your windws host.

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Windows machine.

We will suppose here that your server is named *srv-win-1*. Of course change this name with the real name of your server.

Find your host definition and edit it:

Under Linux:

```
: :
```

linux:~ # vi /etc/shinken/hosts/srv-win-1.cfg

Or Windows:

```
: :
```

c:wordpad c:shinkenetchostssrv-win-1.cfg

All you need it to add the good template for your host. For example for a Hub Transport server:

```
: :
```

define host{ use dc,windows host_name srv-win-1 address srv-win-1.mydomain.com }

- The use dc is the "template" line. It mean that this host will **inherits** properties from this template. * the host_name is the object name of your host. It must be **unique**. * the address is ... the network address of your host :)

What is checked with a domain controller template?

At this point, you configure your host to be checked with a dc and windows templates. What does it means? In addition to the w

- Domain replication

13.3.8 Restarting Shinken

You're done with modifying the Shinken configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.4 Monitoring a DHCP server

Abstract

This document describes how you can monitor a DHCP service.

13.4.1 Introduction

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you follow the quickstart.

13.4.2 Overview

Note: TODO: draw a dhcp diag

Monitoring a DHCP server means ask for a DHCP query and wait for a response from this server. Don't worry, the DHCP confirmation will never be send, so you won't have a DHCP entry for this test.

13.4.3 Steps

There are some steps you'll need to follow in order to monitor a new database machine. They are:

- Allow check_dhcp to run
- Update your server host definition for dhcp monitoring
- Restart the Shinken daemon

13.4.4 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- Some **check_dhcp** commands definition has been added to the "commands.cfg" file.
- An DHCP host template (called "dhcp") has already been created in the "templates.cfg" file.

The above-mentioned config files can be found in the ///etc/shinken/packs/network/services/dhcp* directory (or *c:shinkenetcpacksnetworkservicesdhcp// under windows). You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your DHCP boxes in no time.

Tip: We are supposing here that the DHCP server you want to monitor is named srv-lin-1 and is a linux. Please change the above lines and commands with the real name of your server of course.

13.4.5 Allow check_dhcp to run

The check_dhcp must be run under the root account to send a dhcp call on the network. To do this, you should launch on your shinken server:

```
chown root:root /usr/lib/nagios/plugins/check_dhcp
chmod u+s /usr/lib/nagios/plugins/check_dhcp
```

13.4.6 Declare your host as an dhcp server

All you need to get all the DHCP service checks is to add the *dhcp* template to this host. We suppose you already monitor the OS for this host, and so you already got the host configuration file for it.

Under Linux:

```
::
```

linux:~ # vi /etc/shinken/hosts/srv-lin-1.cfg

Or Windows:

```
::
```

c:wordpad c:shinkenetchostssrv-lin-1.cfg

And add:

```
::
```

define host{ use dhcp,linux host_name srv-lin-1 address srv-lin-1.mydomain.com

}

13.4.7 Restarting Shinken

You're done with modifying the Shiknen configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.5 Monitoring Microsoft Exchange

Abstract

This document describes how you can monitor devices running Microsoft Exchange. This monitoring covers typically:

- Hub transport activity
- Hub transport queues
- Database activities
- Receive/send activities
- etc ...

13.5.1 Introduction

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that was installed if you followed the quickstart.

13.5.2 Overview

Monitoring an Exchange device is possible the agentless by polling via the network using the WMI protocol, like we proposed in the windows template.

13.5.3 Prerequisites

Have a valid account on the Microsoft Windows device (local or domain account) you will monitor using WMI queries and already get the windows server monitor with the windows template.

13.5.4 Steps

There are several steps you'll need to follow in order to monitor a Microsoft Exchange server.

- Add the good exchange template to your windows host in the configuration
- Restart the Shinken Arbiter

13.5.5 What's Already Been Done For You

To make your life a bit easier, configuration templates are provided as a starting point:

- A selection of **check_exchange_*** based commands definitions have been added to the "commands.cfg" file. This allows you to use the **check_wmi_plus** plugin.
- Some Exchange host templates are included the "templates.cfg" file. This allows you to add new host definitions in a simple manner.

The above-mentioned config files can be found in the ///etc/shinken/packs/microsoft/exchange// directory. You can modify the definitions in these and other templates to suit your needs. However, wait until you're more familiar with Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your devices in no time.

13.5.6 Setup the check_wmi_plus plugin

If you already followed the *windows monitoring* tutorial, you should have the check_wmi_plus plugin installed. If it's not, please do it before activating this pack.

13.5.7 Declare your host in Shinken

There are some templates available for the exchange monitoring, each for an exchange role.

- Hub transport: exchange-ht template
- Mail Box server: exchange-mb template
- CAS server: exchange-cas template

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Windows machine.

We will suppose here that your server is named *srv-win-1*. Of course change this name with the real name of your server.

Find your host definition and edit it:

Under Linux:

```
::
```

linux:~ # vi /etc/shinken/hosts/srv-win-1.cfg

Or Windows:

```
::
```

c:wordpad c:shinkenetchostssrv-win-1.cfg

All you need it to add the good template for your host. For example for a Hub Transport server:

```
::
```

define host{ use exchange-ht,exchange,windows host_name srv-win-1 address srv-win-1.mydomain.com }

- The use exchange-ht and exchange is the "template" line. It mean that this host will **inherits** properties from theses templates. * the host_name is the object name of your host. It must be **unique**. * the address is ... the network address of your host :)

The "exchange" template do not add any specific checks, but allow to link with the exchange contacts easily.

What is checked with an exchange template?

At this point, you configure your host to be checked with a windows template. What does it means? In addition to the windows c

- Hub transport activity
- Hub transport queues
- Database activities
- Receive/send activities

The exchange-cas and exchange-mb do not have any specific checks from now.

Note: Any help is welcome here :)

13.5.8 Restarting Shinken

You're done with modifying the Shinken configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.6 Monitoring a IIS server

Abstract

This document describes how you can monitor a IIS server. This monitoring covers typically:

- connections count
- number of errors
- number of logged users
- etc...

13.6.1 Introduction

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that was installed if you followed the quickstart.

13.6.2 Overview

Monitoring a domain controler is possible in an agentless way by polling via the network using the WMI protocol, like we proposed in the windows template.

13.6.3 Prerequisites

Have a valid account on the Microsoft Windows device (local or domain account) you will monitor using WMI queries and already get the windows server monitor with the windows template.

13.6.4 Steps

There are several steps you'll need to follow in order to monitor a Microsoft IIS server.

- Add the good iis template to your windows host in the configuration
- Restart the Shinken Arbiter

13.6.5 What's Already Been Done For You

To make your life a bit easier, configuration templates are provided as a starting point:

- A selection of **check_iis_**** based commands definitions have been added to the "commands.cfg" file. This allows you to use the **check_wmi_plus** plugin.

- A IIS host template is included the "templates.cfg" file. This allows you to add new host definitions in a simple manner.

The above-mentioned config files can be found in the ///etc/shinken/packs/microsoft/iis// directory. You can modify the definitions in these and other templates to suit your needs. However, wait until you're more familiar with Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your devices in no time.

13.6.6 Setup the check_wmi_plus plugin

If you already follow the *windows monitoring* tutorial, you should already got the check_wmi_plus plugin. If it's not done, please do it first.

13.6.7 Declare your host in Shinken

The IIS template name is *iis*. All you need is to add it on your windws host.

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Windows machine.

We will suppose here that your server is named *srv-win-1*. Of course change this name with the real name of your server.

Find your host definition and edit it:

Under Linux:

```
: :
```

linux:~ # vi /etc/shinken/hosts/srv-win-1.cfg

Or Windows:

```
: :
```

c:wordpad c:shinkenetchostssrv-win-1.cfg

All you need it to add the good template for your host. For example for a Hub Transport server:

```
: :
```

define host{ use iis,windows host_name srv-win-1 address srv-win-1.mydomain.com }

- The use iis is the "template" line. It mean that this host will **inherits** properties from this template. * the host_name is the object name of your host. It must be **unique**. * the address is ... the network address of your host :)

What is checked with a IIS template?

At this point, you configure your host to be checked with a IIS and Windows templates. What does it means? In addition to the windows classic checks, it means that you got some checks already configured for you:

- connections count
- number of errors
- number of logged users
- etc ...

13.6.8 Restarting Shinken

You're done with modifying the Shinken configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.7 Monitoring Linux Devices

Abstract

This document describes how you can monitor "private" services and attributes of Linux devices, such as:

- Memory usage
- CPU load
- Disk usage
- Running processes
- etc.

13.7.1 Introduction

Publicly available services that are provided by Linux devices (like "HTTP", "FTP", "POP3", etc.) can be monitored easily by following the documentation on Monitoring *publicly available services (HTTP, FTP, SSH, etc.)*.

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you followed the quickstart.

13.7.2 Overview

Note: TODO: draw a by snmp diag

You can monitor a Linux device with an snmp agent, with a local agent and via SSH.

- This tutorial will focus on the SNMP based method.
- A local agent can provide faster query interval, more flexibility, passive and active communication methods.
- SSH based communications and checks should only be executed for infrequent checks as these have a high impact on the client and server cpu. These also are very slow to execute overall and will not scale when polling thousands of devices.

13.7.3 Steps

Here are the steps you will need to follow in order to monitor a new Linux device:

- Install/configure SNMPd on the Linux device
- Create new host definition for monitoring this device
- Restart the Shinken daemon

13.7.4 What's Already Been Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A selection of **check_snmp_** command definitions has been added to the "commands.cfg" file.
- A Linux host template (called "linux") has already been created in the "templates.cfg" file. This allows you to add new host definitions with a simple keyword.

The above-mentioned configuration files can be found in the ///etc/shinken///packs/os/linux directory (or *c:shinkenetcpacksoslinux* under windows). You can modify the definitions in these and other configuration packs to suit your needs better. However, it is recommended to wait until you're more familiar with Shinken before doing so. For the time being, just follow the directions outlined below and you'll be securely monitoring your Linux boxes in no time.

Tip: In the example, the linux device being monitored is named srv-lin-1. To re-use the example, make sure to update the hostname to that of your server.

13.7.5 Installing/setup snmpd on srv-lin-1

First connect as root under srv-lin-1 with SSH (or putty/SecureCRT under windows).

Note: Todo: check if shinken.sh can do this, or with a deploy command?

RedHat like:

```
yum install snmpd
```

Debian like:

```
apt-get install snmpd
```

Edit the /etc/snmp/snmpd.conf and comment the line:

```
agentAddress  udp:127.0.0.1:161
```

and uncomment the line:

```
agentAddress udp:161,udp6:[::1]:161
```

You can change the SNMP community (password) for your host in the line by changing the default value "public" by what you prefer:

```
rocommunity public
```

Restart the snmpd daemon:

```
sudo /etc/init.d/snmpd restart
```

Test the connection

To see if the keys are working, just launch from your Shinken server. Change the "public" community value with your one:

```
::
```

check_snmp -H srv-lin-1 -o .1.3.6.1.2.1.1.3.0 -C public

It should give you the uptime of the srv-lin-1 server.

13.7.6 Declare your new host in Shinken

If the SNMP community value is a global one you are using on all your hosts, you can configure it in the file /etc/shinken/resource.cfg (or c:shinkenresource.cfg under windows) in the line:

```
$SNMPCOMMUNITYREAD$=public
```

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Linux device.

You can add the new **host** definition in an existing configuration file, but it's a good idea to have one file per host, it will be easier to manage in the future. So create a file with the name of your server.

Under Linux:

```
::
```

linux:~ # vi /etc/shinken/hosts/srv-lin-1.cfg

Or Windows:

```
::
```

c:wordpad c:shinkenetchostssrv-lin-1.cfg

You need to add a new host definition for the Linux device that you're going to monitor. Just copy/paste the above definition Change the "host_name", and "address" fields to appropriate values for this device.

```
::
```

define host{ use linux host_name srv-lin-1 address srv-lin-1.mydomain.com

}

- The use linux is the "template" line. It mean that this host will **inherits** properties from the linux template. * the host_name is the object name of your host. It must be **unique**. * the address is ... the network address of your linux server :)

If you are using a specific SNMP community for this host, you can configure it in the SNMPCOMUNITY host macro like this:

```
::
```

define host{ use linux host_name srv-lin-1 address srv-lin-1.mydomain.com _SNMPCOMMUNITY password

}

To enable disk checking for the host, configure the filesystem macro:

```
::
```

define host{ use linux host_name srv-lin-1 address srv-lin-1.mydomain.com _SNMPCOMMUNITY password _fs /, /var

}

What is checked with a linux template?

At this point, you configure your host to be checked with a linux template. What does it means? It means that you got some chec

- host check each 5 minutes: check with a ping that the server is UP
- check disk spaces
- check load average
- check the CPU usage
- check physical memory and swap usage
- check network interface activities

13.7.7 Restarting Shinken

You're done with modifying the Shiknen configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.8 Monitoring Linux Devices

This document describes how you can monitor "private" services and attributes of GNU/Linux devices, such as:

- Memory usage
- CPU load
- Disk usage
- Running processes
- etc.

13.8.1 Available Methods

Several methods to monitor GNU/Linux devices are available:

- **:ref:'SNMP <monitoring_a_linux_via_snmp>'** – Install or activate the linux SNMP agent and configure it to serve system statistics;
- **:ref:'Local Agent <monitoring_a_linux_via_local_agent>'** – Provides faster query interval, more flexibility, passive and active communication methods;
- **SSH** – Should only be executed for infrequent checks as these have a high impact on the client and server CPU. It's also very slow to execute overall, and will not scale when polling thousands of devices;
- **:ref:'Monitoring Publicly Available Services <monitoring_a_network_service>'** – Public services provided by GNU/Linux devices (like HTTP, FTP, POP3, etc.) can be easily monitored.

13.9 Monitoring a Linux via a Local Agent

Note: **TODO** Documentation needs to be written

See:

- *Chapter 9. Monitoring GNU/Linux & Unix Machines*

13.9.1 NRPE

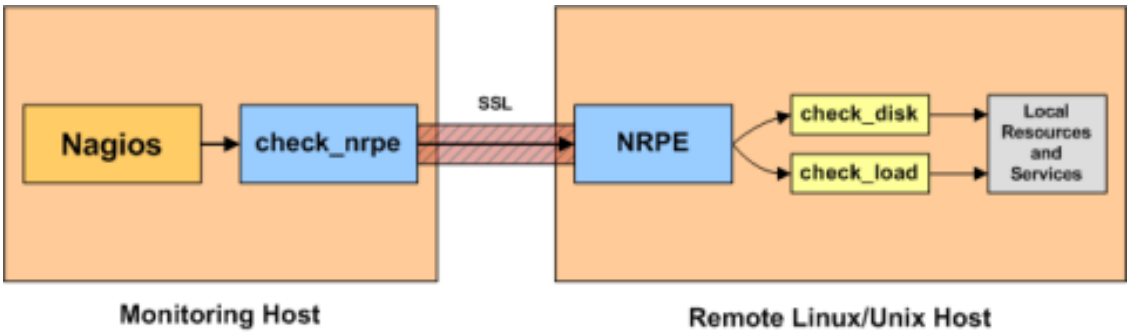

See:

- Shinken Module to bypass check_nrpe tool: *NRPE Module*

13.9.2 NSCA

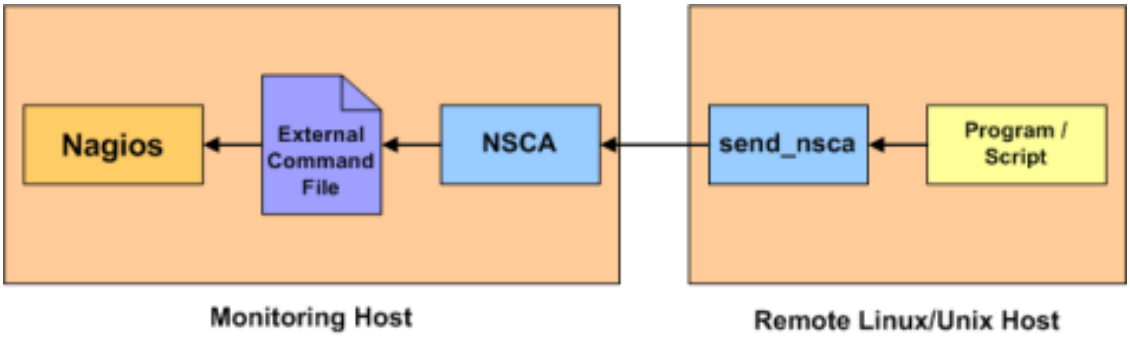

See:

- Shinken Module to natively receive NSCA messages : :ref:‘ <nsca_daemon_module>‘

13.10 Monitoring a Linux via SNMP

Note: TODO: draw a by snmp diag

Instructions below assume that you've installed Shinken according to the *10 Minutes Installation Guide*. The sample configuration entries below reference objects that are defined in the sample config files (‘’commands.cfg’‘, ‘’templates.cfg’‘, etc.) installed if you followed this quickstart.

13.10.1 Steps

Here are the steps you will need to follow in order to monitor a new GNU/Linux device:

- Install/configure SNMPd on the GNU/Linux device
- Create new host definition for monitoring this device
- Restart the Shinken daemon

13.10.2 What's Already Been Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A selection of **check_snmp_** command definitions has been added to the "commands.cfg" file.
- A Linux host template (called "linux") has already been created in the "templates.cfg" file. This allows you to add new host definitions with a simple keyword.

The above-mentioned configuration files can be found in the ///etc/shinken///packs/os/linux directory (or *c:shinkenetcpacksoslinux* under windows). You can modify the definitions in these and other configuration packs to suit your needs better. However, it is recommended to wait until you're more familiar with Shinken before doing so. For the time being, just follow the directions outlined below and you'll be securely monitoring your Linux boxes in no time.

Tip: In the example, the linux device being monitored is named srv-lin-1. To re-use the example, make sure to update the hostname to that of your server.

13.10.3 Installing/setup snmpd on srv-lin-1

First connect as root under srv-lin-1 with SSH (or putty/SecureCRT under windows).

Note: Todo: check if shinken.sh can do this, or with a deploy command?

RedHat like:

```
yum install snmpd
```

Debian like:

```
apt-get install snmpd
```

Edit the /etc/snmp/snmpd.conf and comment the line:

```
agentAddress  udp:127.0.0.1:161
```

and uncomment the line:

```
agentAddress udp:161,udp6:[::1]:161
```

You can change the SNMP community (password) for your host in the line by changing the default value "public" by what you prefer:

```
rocommunity public
```

Restart the snmpd daemon:

```
sudo /etc/init.d/snmpd restart
```

Test the connection

To see if the keys are working, just launch from your Shinken server. Change the "public" community value with your one:

```
::
```

check_snmp -H srv-lin-1 -o .1.3.6.1.2.1.1.3.0 -C public

It should give you the uptime of the srv-lin-1 server.

13.10.4 Declare your new host in Shinken

If the SNMP community value is a global one you are using on all your hosts, you can configure it in the file /etc/shinken/resource.cfg (or c:shinkenresource.cfg under windows) in the line:

```
$SNMPCOMMUNITYREAD$=public
```

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Linux device.

You can add the new **host** definition in an existing configuration file, but it's a good idea to have one file per host, it will be easier to manage in the future. So create a file with the name of your server.

Under Linux:

```
::
```

linux:~ # vi /etc/shinken/hosts/srv-lin-1.cfg

Or Windows:

```
::
```

c:wordpad c:shinkenetchostssrv-lin-1.cfg

You need to add a new host definition for the GNU/Linux device that you're going to monitor. Just copy/paste the above definition Change the **host_name** and **address** fields to appropriate values for this device.

```
::
```

define host{ use linux host_name srv-lin-1 address srv-lin-1.mydomain.com

}

- The use linux is the "template" line. It mean that this host will **inherits** properties from the linux template. * the host_name is the object name of your host. It must be **unique**. * the address is ... the network address of your linux server :)

If you are using a specific SNMP community for this host, you can configure it in the SNMPCOMUNITY host macro like this:

```
::
```

define host{ use linux host_name srv-lin-1 address srv-lin-1.mydomain.com _SNMPCOMMUNITY password

}

What is checked with a linux template ?

At this point, you configure your host to be checked with a linux template. What does it means? It means that you got some chec

- host check each 5 minutes: check with a ping that the server is UP
- check disk spaces
- check load average
- check the CPU usage
- check physical memory and swap usage
- check network interface activities

13.10.5 Restarting Shinken

You're done with modifying the Shinken configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any error messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.11 Monitoring Publicly Available Services

Abstract

This document describes how you can monitor publicly available services, applications and protocols. By "public" I mean services that are accessible across the network - either the local network or Internet. Examples of public services include "HTTP", "POP3", "IMAP", "FTP", and "SSH". There are many more public services that you probably use on a daily basis. These services and applications, as well as their underlying protocols, can usually be monitored by Shinken without any special access requirements.

13.11.1 Introduction

Private services, in contrast, cannot be monitored with Shinken without an intermediary agent of some kind. Examples of private services associated with hosts are things like CPU load, memory usage, disk usage, current user count, process information, etc.

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you followed the quickstart.

13.11.2 Plugins For Monitoring Services

When you find yourself needing to monitor a particular application, service, or protocol, chances are good that a *plugin* exists to monitor it. The official Nagios/Shinken plugins distribution comes with plugins that can be used to monitor a variety of services and protocols. There are also a large number of contributed plugins that can be found in the "contrib/" subdirectory of the plugin distribution. The Monitoringexchange.org website hosts a number of additional plugins that have been written by users, so check it out when you have a chance.

If you don't happen to find an appropriate plugin for monitoring what you need, you can always write your own. Plugins are easy to write, so don't let this thought scare you off. Read the documentation on *developing plugins* for more information.

I'll walk you through monitoring some basic services that you'll probably use sooner or later. Each of these services can be monitored using one of the plugins that gets installed as part of the Nagios/Shinken plugins distribution. Let's get started...

13.11.3 The host definition

Before you can monitor a service, you first need to define a host that is associated with the service. If you follow the windows/linux/printer monitoring tutorial, you should be familiar with the process of creating a host and linking your services to it.

For this example, lets say you want to monitor a variety of services on a remote windows host. Let's call that host *srv-win-1*. The host definition can be placed in its own file. Here's what the host definition for *remotehost* might look like if you followed the windows monitoring tutorial:

```
::

    define host{ use windows host_name srv-win-1 address srv-win-1.mydomain.com

    }
```

13.11.4 Classic services definition with templates

For classic services like HTTP(s) or FTP, there are some ready to run template that you can use. The full service definition is explained in another tutorial Services definitions.

The idea here is just to *tag* your host with what it is providing as network services and automatically get some default checks like Http or Ftp ones.

13.11.5 Monitoring HTTP

Chances are you're going to want to monitor web servers at some point - either yours or someone else's. The **check_http** plugin is designed to do just that. It understands the "HTTP" protocol and can monitor response time, error codes, strings in the returned HTML, server certificates, and much more.

There is already a *http* template for your host that will create for you a Http service. All you need is to add this template on your host, with a comma for separating templates.

So you host definition will now look like:

```
::

    define host{ use windows,http host_name srv-win-1 address srv-win-1.mydomain.com

    }
```

Note: TODO: write some custom macros for other page check or timeout

It will create a Http service that will look for the "/" page running on *srv-win-1*. It will produce alerts if the web server doesn't respond within 10 seconds or if it returns "HTTP" errors codes (403, 404, etc.). That's all you need for basic monitoring. Pretty simple, huh?

13.11.6 Monitoring HTTPS

We got more an more HTTPS services. You will basically check two things: page accessibility and certificates.

There is already a *https* template for your host that will create for you a Https and a HttpsCertificate services. The Https check is like the Http one, but on the SSL port. The HttpsCertificate will look for the expiration of the certificate, and will warn you 30 days before the end of the certificate, and raise a critical alert if its expired.

So you host definition will now look like:

```
::
```

define host{ use windows,https host_name srv-win-1 address srv-win-1.mydomain.com

}

Note: TODO: write some custom macros for other page check or timeout

You can check Http AND Https in the same time, all you need is to use the two templates in the same time:

```
::
```

define host{ use windows,http,https host_name srv-win-1 address srv-win-1.mydomain.com

}

13.11.7 Monitoring FTP

When you need to monitor "FTP" servers, you can use the **check_ftp** plugin. Like for the Http case, there is already a ftp template that you can use.

```
::
```

define host{ use windows,ftp host_name srv-win-1 address srv-win-1.mydomain.com

}

This service definition will monitor the "FTP" service and generate alerts if the "FTP" server doesn't respond within 10 seconds.

13.11.8 Monitoring SSH

When you need to monitor "SSH" servers, you can use the **check_ssh** plugin.

```
::
```

define host{ use windows,ssh host_name srv-win-1 address srv-win-1.mydomain.com

}

Tip: You don't need to declare the ssh template if you already configure your host with the linux one, the Ssh service is already configured.

This definition will monitor the "Ssh" service and generate alerts if the "SSH" server doesn't respond within 10 seconds.

13.11.9 Monitoring SMTP

The **check_smtp** plugin can be using for monitoring your email servers. You can use the smtp template for you host to automatically get a Smtp service check.

```
::
```

define host{ use windows,smtp host_name srv-win-1 address srv-win-1.mydomain.com

}

This service definition will monitor the "Smtp" service and generate alerts if the "SMTP" server doesn't respond within 10 seconds.

13.11.10 Monitoring POP3

The **check_pop** plugin can be using for monitoring the "POP3" service on your email servers. Use the *pop3* template for your host to get automatically a Pop3 service.

```
::
```

define host{ use windows,pop3 host_name srv-win-1 address srv-win-1.mydomain.com

}

This service definition will monitor the "POP3" service and generate alerts if the "POP3" server doesn't respond within 10 seconds.

13.11.11 Monitoring IMAP

The **check_imap** plugin can be using for monitoring "IMAP4" service on your email servers. You can use the *imap* template for your host to get an Imap service check.

```
::
```

define host{ use windows,imap host_name srv-win-1 address srv-win-1.mydomain.com

}

This service definition will monitor the "IMAP4" service and generate alerts if the "IMAP" server doesn't respond within 10 seconds.

To get smtp, pop3 and imap service checks, you can just link all theses templates to your host:

```
::
```

define host{ use windows,smtp,pop3,imap host_name srv-win-1 address srv-win-1.mydomain.com

}

13.11.12 Restarting Shinken

Once you've added the new host templates to your object configuration file(s), you're ready to start monitoring them. To do this, you'll need to *verify your configuration* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.12 Monitoring a printer

Abstract

This document describes how you can monitor a printer. Specifically, HP™ printers that have internal/external JetDirect® cards/devices, or other print servers (like the Troy™ PocketPro 100S® or the Netgear™ PS101®) that support the JetDirect protocol.

The **check_hpjd** plugin (which is part of the standard Nagios/Shinken plugins distribution) allows you to monitor the status of JetDirect-capable printers which have "SNMP" enabled. The plugin is capable of detecting the following printer states:

- Paper Jam
- Out of Paper
- Printer Offline
- Intervention Required
- Toner Low
- Insufficient Memory
- Open Door
- Output Tray is Full
- and more...

13.12.1 Introduction

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you follow the quickstart.

13.12.2 Overview

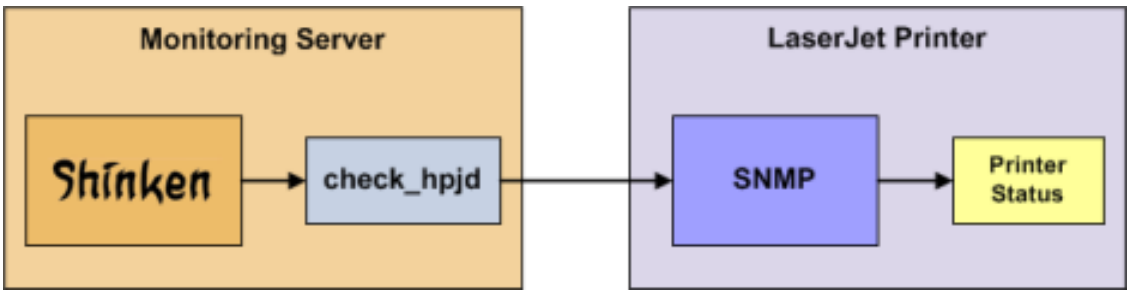

Monitoring the status of a networked printer is pretty simple. JetDirect-enabled printers usually have "SNMP" enabled, which allows Shinken to monitor their status using the **check_hpjd** plugin.

The **check_hpjd** plugin will only get compiled and installed if you have the net-snmp and net-snmp-utils packages installed on your system. Make sure the plugin exists in "/var/lib/nagios/" before you continue. If it doesn't, install net-snmp and net-snmp-utils and recompile/reinstall the Nagios plugins.

13.12.3 Steps

There are some steps you'll need to follow in order to monitor a new printer machine. They are:

- Create new host definition for monitoring this machine
- Restart the Shinken daemon

13.12.4 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A *check_hpjd* command definition has been added to the "commands.cfg" file. This allows you to use the **check_hpjd** plugin to monitor network printers.
- A printer host template (called *printer*) has already been created in the "templates.cfg" file. This allows you to add new printer host definitions in a simple manner.

The above-mentioned config files can be found in the ///etc/shinken/// directory (or *c:shinkenetc* under windows). You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your printers in no time.

Tip: We are supposing here that the printer machine you want to monitor is named printer-1. Please change the above lines and commands with the real name of your printer of course.

13.12.5 Declare your new printer in Shinken

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Linux machine.

You can add the new **host** definition in an existing configuration file, but it's a good idea to have one file by host, it will be easier to manage in the future. So create a file with the name of your server.

Under Linux:

```
: :
```

linux:~ # vi /etc/shinken/hosts/printer-1.cfg

Or Windows:

```
: :
```

c:wordpad c:shinkenetchostsprinter-1.cfg

You need to add a new host definition for the Linux machine that you're going to monitor. Just copy/paste the above definition Change the "host_name", and "address" fields to appropriate values for this machine.

```
: :
```

define host{ use printer host_name printer-1 address 192.160.0.1

}

- The use printer is the "template" line. It mean that this host will **inherits** properties from the printer template. * the host_name is the object name of your host. It must be **unique**. * the address is ... the network address of your printer. It can be a FQDN or an IP. :)

What is checked with a printer template ?

At this point, you configure your host to be checked with a printer template. What does it means? It means that you got some ch

- printer check each 5 minutes: check with a ping that the printer is UP

Note: TODO: fill what is checked with HPJD

13.12.6 Restarting Shinken

You're done with modifying the Shinken configuration, so you will need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.13 Monitoring Network Devices

Abstract

This document describes how you can monitor network devices (Cisco, Nortel, Procurve,...), such as:

- Network usage
- CPU load
- Memory usage
- Port state
- Hardware state
- etc.

13.13.1 Introduction

These instructions assume that you have installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you followed the quickstart.

13.13.2 Overview

Note: TODO: draw a by snmp diag

Network devices are typically monitored using the SNMP and ICMP (ping) protocol.

13.13.3 Steps

Here are the steps you will need to follow in order to monitor a new device:

- Setup check_nwc_health and try a connection with the equipment
- Create new host definition to monitor this device
- Restart the Shinken daemon

13.13.4 What's Already Been Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A selection of **check_nwc_health** command definitions have been added to the "commands.cfg" file.
- A network equipment host template (called "switch") has already been created in the "templates.cfg" file. This allows you to add new host definitions with a simple keyword.

The above-mentioned configuration files can be found in the ///etc/shinken///packs/network/switch directory (or *c:shinkenetcpacksnetworkswitch* under windows). You can modify the definitions in these and other configuration packs to suit your needs better. However, it is recommended to wait until you are familiar with Shinken before doing so. For the time being, just follow the directions outlined below and you will be securely monitoring your devices in no time.

Tip: In the example, the switch device being monitored is named switch-1. To re-use the example, make sure to update the hostname to that of your device.

13.13.5 Setup check_nwc_health and try a connection switch-1

First connect as the shinken user under your shinken host.

Unix like to install check_nwc_health:

```
install -p check_nwc_health
```

Now to try to check your switch, for this you need a read community for it. Consult your device vendors documentation on how to change the SNMP community. The default value is "public". The most efficient, though less secure protocol version of SNMP is version 2c. Version 3 includes encryption and user/password combinations, but is more convoluted to configure and may tax your devices CPU, it is beyond the scope of this tutorial.

Now connect as the shinken user.

```
su - shinken
```

Warning: NEVER launch plugins like check_* under the root account, because it can work under root but will deposit temporary files that will break the plugins when executed with the shinken user.

Let's say that the switch-1 IP is 192.168.0.1.

```
/usr/local/shinken/libexec/check_nwc_health --hostname 192.168.0.1 --timeout 60 --community "public"
```

It should give you the state of all interfaces.

13.13.6 Declare your switch in Shinken

If the SNMP community value is a global one you are using on all your hosts, you can configure it in the file /etc/shinken/resource.cfg (or c:shinkenresource.cfg under windows) in the line:

```
$SNMPCOMMUNITYREAD$=public
```

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Linux device.

You can add the new **host** definition in an existing configuration file, but it's a good idea to have one file per host, it will be easier to manage in the future. So create a file with the name of your server.

Under Linux:

```
::
```

linux:~ # vi /etc/shinken/hosts/switch-1.cfg

Or Windows:

```
::
```

c:wordpad c:shinkenetchostsswitch-1.cfg

You need to add a new host definition for the switch device that you're going to monitor. Just copy/paste the above definition Change the "host_name", and "address" fields to appropriate values for this device.

```
::
```

define host{ use switch host_name switch-1 address 192.168.0.1

}

- The use switch is the "template" line. It mean that this host will **inherit** properties and checks from the switch template. * the host_name is the object name of your host. It must be **unique**. * the address is ... the network address of your switch :)

If you are using a specific SNMP community for this host, you can configure it in the SNMPCOMUNITY host macro like this:

```
::
```

define host{ use switch host_name switch-1 address 192.168.0.1 _SNMPCOMMUNITY password

}

What is checked with a switch template ?

At this point, you configure your host to be checked with a switch template. What does it means? It means that you got some che

- host check each 5 minutes: check with a ping that the device is UP
- interface usage
- interface status
- interface errors

For CPU/memory/Hardware checks?

Not all devices are managed by check_nwc_health. To know if yours is, just launch:

```
/usr/local/shinken/libexec/check_nwc_health --hostname 192.168.0.1 --timeout 60 --community "public"
```

If it's ok, you can add the "cisco" template for your hosts (even if it's not a cisco device, we are working on getting more templates configuration).

```
::
```

define host{ use cisco,switch host_name switch-1 address 192.168.0.1 _SNMPCOMMUNITY password

}

If it does not work, to learn more about your device, please launch the command:

```
snmpwalk -v2c -c public 192.168.0.1 | bzip2 > /tmp/device.bz2
```

And launch this this command as well:

```
nmap -T4 -O -oX /tmp/device.xml 192.168.0.1
```

Once you have done that, send us the device.bz2 and device.xml files (located in /tmp directory), we will add this new device to the check_nwc_health plugin in addition to the discovery module. With these files please also provide some general information about the device, so we will incorporate it correctly into the discovery module.

13.13.7 Restarting Shinken

You're done with modifying the Shinken configuration, you will need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.14 Monitoring Windows Devices

Abstract

This document describes how you can monitor devices running Microsoft Windows using a predefined template. This template can address:

- Memory usage
- CPU load
- Disk usage
- Service states
- Running processes
- Event logs (Application or system)
- etc.

13.14.1 Introduction

Publicly available services that are provided by Windows machines ("HTTP", "FTP", "POP3", etc.) can be monitored by following the documentation on *Monitoring publicly available services (HTTP, FTP, SSH, etc.)*.

The instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that was installed if you followed the quickstart.

13.14.2 Overview

Monitoring a windows device is possible using two different methods:

- Agent: by installing software such as NSClient++
- Agentless: by polling via the network using the WMI protocol

This document focuses on the agentless method. The agent based method is described in windows monitoring with nsclient++ <windows monitoring with nsclient++>.

13.14.3 Prerequisites

Have a valid account on the Microsoft Windows device (local or domain account) you will monitor using WMI queries.

13.14.4 Steps

There are several steps you'll need to follow in order to monitor a Microsoft Windows device.

- Install check_wmi_plus plugin
- Setup a account on the server for the WMI queries
- Declare your windows host in the configuration
- Restart the Shinken Arbiter

13.14.5 What's Already Been Done For You

To make your life a bit easier, configuration templates are provided as a starting point:

- A selection of **check_windows** based command definitions have been added to the "commands.cfg" file. This allows you to use the **check_wmi_plus** plugin.
- A Windows host template (called "windows") is included the "templates.cfg" file. This allows you to add new Windows host definitions in a simple manner.

The above-mentioned config files can be found in the ///etc/shinken/packs/os/windows// directory. You can modify the definitions in these and other templates to suit your needs. However, wait until you're more familiar with Shinken before doing so. For the time being, just follow the directions outlined below and you will be monitoring your Windows devices in no time.

13.14.6 Setup the check_wmi_plus plugin

The plugin used for windows agent less monitoring is check_wmi_plus. To install it, just launch as root on your shinken server:

```
./install -p check_wmi_plus
```

(install is a executable script in the git, the previous name was shinken.sh)

13.14.7 Setup a windows account for WMI queries

TODO: write on using less than server admin

You need to configure your user account int the /etc/shinken/resources.cfg file or the c:shinkenetcresource.cfg file under windows with the one you just configured:

```
$DOMAINUSER$=shinken_user
$DOMAINPASSWORD$=superpassword
```

13.14.8 Declare your host in Shinken

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Windows device.

We will assume that your server is named *srv-win-1*. Replace this with the real hostname of your server.

You can add the new **host** definition in an existing configuration file, but it is good practice to have one file per host, it will be easier to manage in the future. So create a file with the name of your server.

Under Linux:

```
::
```

linux:~ # vi /etc/shinken/hosts/srv-win-1.cfg

Or Windows:

```
::
```

c:wordpad c:shinkenetchostssrv-win-1.cfg

You need to add a new host definition for the Windows device that you will monitor. Just copy/paste the above definition, change the "host_name", and "address" fields to appropriate values.

```
::
```

define host{ use windows host_name srv-win-1 address srv-win-1.mydomain.com }

- use windows is the "template" line. This host will **inherit** properties from the windows template. * host_name is the object name of your host. It must be **unique**. * address is the ip address or hostname of your host (FQDN or just the host portion).

Note: If you use a hostname be aware that you will have a DNS dependency in your monitoring system. Either have a periodically updated local hosts file with all relevant entries, long caching or use an IP address.

What is checked by the windows template ?

You have configured your host to be checked by the windows template. What does it means? It means that you got some checks

- host check each 5 minutes with a ping
- check disk spaces
- check if autostarting services are started
- check CPU load (total and each CPU)
- check memory and swap usage
- check for a recent (less than one hour) reboot
- critical/warnings errors in the application and system event logs
- too much inactive RDP sessions
- CPU hog processes

13.14.9 Restarting Shinken

You're done with modifying the Shinken configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.15 Monitoring Microsoft Mssql server

Abstract

This document describes how you can monitor a Mssql server such as:

- Connection time
- A recent restart
- The number of connections
- Cache hit
- Dead locks
- etc...

13.15.1 Introduction

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you follow the quickstart.

13.15.2 Overview

Note: TODO: draw a check_mssql diag

Monitoring a Mssql server need the plugin check_mssql_health available at **'labs.consol.de/lang/en/nagios/check_mssql_health/'_** and a mssql user account for the connection.

13.15.3 Steps

There are some steps you'll need to follow in order to monitor a new database machine. They are:

- Install check plugins
- setup the mssql user account
- Update your windows server host definition for mysql monitoring
- Restart the Shinken daemon

13.15.4 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- Some **check_mssql_** commands definition has been added to the "commands.cfg" file.
- A Mssql host template (called "mssql") has already been created in the "templates.cfg" file.

The above-mentioned config files can be found in the ///etc/shinken/// directory (or *c:shinkenetc* under windows). You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your Mssql boxes in no time.

Tip: We are supposing here that the Mssql machine you want to monitor is named srv-win-1. Please change the above lines and commands with the real name of your server of course.

13.15.5 Installing the check plugins on Shinken

First connect as root under you Shinken server (or all poller servers for a multi-box setup)

Note: Todo: Use shinken.sh for this

13.15.6 Setup the mysql user account

Look at the **'labs.consol.de/lang/en/nagios/check_mssql_health/'_** page about how to conigure your user connection.

Then you will need to configure your user/password in the macros file so the plugins will have the good values for the connction. So update the /etc/shinken/resource.cfg file or c:shinkenetcresource.cfg file to setup the new password:

```
$MSSQLUSER$=shinken
$MSSQLPASSWORD$=shinkenpassword
```

Test the connection

To see if the connection is ok, just launch:

```
::
```

> /var/lib/nagios/plugins/check_mssql_health –hostname srv-win-1 –username shinken –password shinken-password –mode connection-time

It should not return errors.

13.15.7 Declare your host as a mssql server

All you need to get all the Msql service checks is to add the *mssql* template to this host. We suppose you already monitor the OS for this host, and so you already got the host configuration file for it.

Under Linux:

```
: :
```

linux:~ # vi /etc/shinken/hosts/srv-win-1.cfg

Or Windows:

```
: :
```

c:wordpad c:shinkenetchostssrv-win-1.cfg

You need to add the mysql template in the use line. It's better to follow the more precise template to the less one, like here mssql first, and then windows.

```
: :
```

define host{ use mssql,windows host_name srv-win-1 address srv-win-1.mydomain.com

}

What is checked with a mssql template ?

At this point, you configure your host to be checked with a mssql template. What does it means? It means that you got some ser

- connection-time : Measures how long it takes to login 0..n seconds (1, 5)
- connected-users : Number of connected users 0..n (50, 80)
- cpu-busy : CPU Busy Time 0%..100% (80, 90)
- io-busy : IO Busy Time 0%..100% (80, 90)
- full-scans : Number of Full Table Scans per second 0..n (100, 500)
- transactions : Number of Transactions per second 0..n (10000, 50000)
- batch-requests : Number of Batch Requests per second 0..n (100, 200)
- latches-waits : Number of Latch-Requests per second, which could not be fulfilled 0..n (10, 50)
- latches-wait-time : Average time a Latch-Request had to wait until it was granted 0..n ms (1, 5)
- locks-waits : Number of Lock-Requests per second, which could not be satisfied. 0..n (100, 500)
- locks-timeouts : Number of Lock-Requests per second, which resulted in a timeout. 0..n (1, 5)
- locks-deadlocks : Number of Deadlocks per second 0..n (1, 5)
- sql-recompilations : Number of Re-Compilations per second 0..n (1, 10)
- sql-initcompilations : Number of Initial Compilations per second 0..n (100, 200)
- total-server-memory : The main memory reserved for the SQL Server 0..n (nearly1G, 1G)
- mem-pool-data-buffer-hit-ratio : Data Buffer Cache Hit Ratio 0%..100% (90, 80:)
- lazy-writes : Number of Lazy Writes per second 0..n (20, 40)
- page-life-expectancy : Average time a page stays in main memory 0..n (300:, 180:)

- free-list-stalls : Number of Free List Stalls per second 0..n (4, 10)
- checkpoint-pages : Number of Flushed Dirty Pages per second 0..n ()
- database-free : Free space in a database (Default is percent, but –units can be used also). You can select a single database with the name parameter. 0%..100% (5%, 2%)
- database-backup-age Elapsed time since a database was last backupped (in hours). The performancedata also cover the time needed for the backup (in minutes). 0..n

13.15.8 Restarting Shinken

You're done with modifying the Shiknen configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors! .. _labs.consol.de/lang/en/nagios/check_mssql_health/: http://labs.consol.de/lang/en/nagios/check_mssql_health/

13.16 Monitoring MySQL

Abstract

This document describes how you can monitor a MySQL server such as:

- Connection time
- A recent restart
- The number of connections
- Cache hit
- etc...

13.16.1 Introduction

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you follow the quickstart.

13.16.2 Overview

Note: TODO : draw a check_mysql diag

Monitoring a MySQL server need the plugin check_mysql_health available at **'labs.consol.de/lang/en/nagios/check_mysql_health/'_** and a mysql user account for the connection.

13.16.3 Steps

There are some steps you'll need to follow in order to monitor a new Linux machine. They are:

- Install check plugins

- setup the mysql user account
- Update your server host definition for mysql monitoring
- Restart the Shinken daemon

13.16.4 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- Some **check_mysql_** commands definition has been added to the "commands.cfg" file.
- A Mysql host template (called "mysql") has already been created in the "templates.cfg" file.

The above-mentioned config files can be found in the ///etc/shinken/// directory (or *c:shinkenetc* under windows). You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your Mysql boxes in no time.

Tip: We are supposing here that the Mysql machine you want to monitor is named srv-lin-1 and is a Linux. Please change the above lines and commands with the real name of your server of course.

13.16.5 Installing the check plugins on Shinken

First connect as root under you Shinken server (or all poller servers for a multi-box setup) and launch :

```
shinken.sh -p check_mysql_health
```

13.16.6 Setup the mysql user account

Connect with a root account on your MySQL database. change 'password' with your mysql root password :

```
lin-srv-1:# mysql -u root -ppassword
```

And create a shinken user :

```
GRANT usage ON *.* TO 'shinken'@'%' IDENTIFIED BY 'shinkenpassword';
```

It's a good thing to change the shinkenpassword to another password. Then you need to update the /etc/shinken/resource.cfg file or c:shinkenetcresource.cfg file to setup the new password :

```
$MYSQLUSER$=shinken
$MYSQLPASSWORD$=shinkenpassword
```

Test the connection

To see if the connection is ok, just launch :

```
::
```

> /var/lib/nagios/plugins/check_mysql_health –hostname srv-lin-1 –username shinken –password shinken-password –mode connection-time

It should not return errors.

13.16.7 Declare your host as a mysql server

All you need to get all the MySQL service checks is to add the *mysql* template to this host. We suppose you already monitor the OS (linux or windows for example) for this host, and so you already got the host configuration file for it.

Under Linux :

```
: :
```

linux:~ # vi /etc/shinken/hosts/srv-lin-1.cfg

Or Windows :

```
: :
```

c:wordpad c:shinkenetchostssrv-lin-1.cfg

You need to add the mysql template in the use line. It's better to follow the more precise template to the less one, like here mysql first, and then linux.

```
: :
```

define host{ use mysql,linux host_name srv-lin-1 address srv-lin-1.mydomain.com

}

What is checked with a mysql template ?

At this point, you configure your host to be checked with a mysql template. What does it means? It means that you got some services checks already configured for you. Warning and alert levels are between () :

- connection-time : Determines how long connection establishment and login take, 0..n Seconds (1, 5)
- uptime : Time since start of the database server (recognizes DB-Crash+Restart), 0..n Seconds (10:, 5: Minutes)
- threads-connected : Number of open connections, 1..n (10, 20)
- threadcache-hitrate : Hitrate in the Thread-Cache 0%..100% (90:, 80:)
- querycache-hitrate : Hitrate in the Query Cache 0%..100% (90:, 80:)
- querycache-lowmem-prunes : Displacement out of the Query Cache due to memory shortness n/sec (1, 10)
- keycache-hitrate : Hitrate in the Myisam Key Cache 0%..100% (99:, 95:)
- bufferpool-hitrate : Hitrate in the InnoDB Buffer Pool 0%..100% (99:, 95:)
- bufferpool-wait-free : Rate of the InnoDB Buffer Pool Waits 0..n/sec (1, 10)
- log-waits : Rate of the InnoDB Log Waits 0..n/sec (1, 10)
- tablecache-hitrate : Hitrate in the Table-Cache 0%..100% (99:, 95:)
- table-lock-contention : Rate of failed table locks 0%..100% (1, 2)
- index-usage : Sum of the Index-Utilization (in contrast to Full Table Scans) 0%..100% (90:, 80:)
- tmp-disk-tables : Percent of the temporary tables that were created on the disk instead in memory 0%..100% (25, 50)
- slow-queries : Rate of queries that were detected as "slow" 0..n/sec (0.1, 1)
- long-running-procs : Sum of processes that are running longer than 1 minute 0..n (10, 20)
- slave-lag : Delay between Master and Slave 0..n Seconds
- slave-io-running : Checks if the IO-Thread of the Slave-DB is running

- slave-sql-running : Checks if the SQL-Thread of the Slave-DB is running
- open-files : Number of open files (of upper limit) 0%..100% (80, 95)
- cluster-ndb-running : Checks if all cluster nodes are running.

13.16.8 Restarting Shinken

You're done with modifying the Shiknen configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors! .. _labs.consol.de/lang/en/nagios/check_mysql_health/: http://labs.consol.de/lang/en/nagios/check_mysql_health/

13.17 Monitoring VMware Machines

Abstract

This document describes how you can monitor "private" services and attributes of ESX and VM machines, such as:

- Memory usage
- CPU load
- Disk usage
- Network usage
- etc.

13.17.1 Introduction

In a VMware server we should monitor:

- ESX hosts
- VM started on it

Note: It is a good practice to automatically create dependencies between them so if an ESX goes down, you won't get useless notifications about all VMs on it. Consult the tutorial about the *vmware arbiter module* for more information on this topic..

For theses checks you will need the check_esx3.pl plugin. .. note:: TODO: write in the setup phase about installing it

---

Note: TODO: draw a by vSphere check

13.17.2 Steps

There are some steps you'll need to follow in order to monitor a new vmware esx/vm machine. They are:

- Create an account on the vsphere console server
- Configure your vsphere server in Shinken

- Create new host definition for monitoring an esx server
- Create new host definition for monitoring a vm machine
- Restart the Shinken daemon

13.17.3 What's already done for you

To make your life a bit easier, a few configuration tasks have already been done for you:

- Some **check_esx_** commands definition has been added to the "commands.cfg" file.
- A VMware ESX host template (called "esx") has already been created in the "templates.cfg" file. This allows you to add new host definitions in a simple manner.
- A VMware virtual machine host template (called "vm") has already been created in the templates.cfg file.

The above-mentioned config files can be found in the ///etc/shinken/// directory (or *c:shinkenetc* under windows). You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your VMware boxes in no time.

Tip: We are supposing here that the esx machine you want to monitor is named srv-esx-1 and the virtual machine is a windows vm named srv-vm-1. Please change the above lines and commands with the real name of your server of course.

13.17.4 Create a vSphere account for Shinken

Please ask your VMware administrator to create a Shinken account (can be a windows domain account) and give it read credential on the vphere environnement (on the root datacenter definition with inheritance).

13.17.5 Setup the vSphere console server connection

You need to configure in the "resource.cfg" file ("/etc/shinken/resource.cfg" under linux or "c:shinkenetcresource.cfg" under wi

- the VSPHERE host connection
- the login for the connection
- the password

```
#### vSphere (ESX) part
$VCENTER$=vcenter.mydomain.com
$VCENTERLOGIN$=someuser
$VCENTERPASSWORD$=somepassowrd
```

You can then try the connection if you already know about an esx host, like for example myesx:

```
::
```

/var/lib/nagios/plugins/check_esx3.pl -D vcenter.mydomain.com -H myesx -u someuser -p somepassword -l cpu

Deploy the public keys on the linux host

When you got the public/private keys, you can deploy the public key to you linux servers.

```
ssh-copy-id  -i /home/shinken/.ssh/id_rsa.pub shinken@srv-lin-1
```

Tip: Change srv-lin-1 with the name of your server.

Test the connection

To see if the keys are working, just launch:

```
::
```

check_by_ssh -H srv-lin-1 -C uptime

It should give you the uptime of the srv-lin-1 machine.

13.17.6 Declare your new host in Shinken

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Linux machine.

You can add the new **host** definition in an existing configuration file, but it's a good idea to have one file by host, it will be easier to manage in the future. So create a file with the name of your server.

Under Linux:

```
::
```

linux:~ # vi /etc/shinken/hosts/srv-lin-1.cfg

Or Windows:

```
::
```

c:wordpad c:shinkenetchostssrv-lin-1.cfg

You need to add a new host definition for the Linux machine that you're going to monitor. Just copy/paste the above definition Change the "host_name", and "address" fields to appropriate values for this machine.

```
::
```

define host{ use linux host_name srv-lin-1 address srv-lin-1.mydomain.com

}

- The use linux is the "template" line. It mean that this host will **inherits** properties from the linux template. * the host_name is the object name of your host. It must be **unique**. * the address is ... the network address of your linux server :)

What is supervised by the linux template ?

You have configured your host to the checks defined from the linux template. What does this mean? It means that you have som

- host check each 5 minutes: check with a ping that the server is UP
- check disk space

- check if ntpd is started
- check load average
- check physical memory and swap usage
- check for a recent (less than one hour) reboot

13.17.7 Restarting Shinken

You're done with modifying the configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.18 Monitoring Microsoft Mssql server

Abstract

This document describes how you can monitor a Mssql server such as:

- Connection time
- A recent restart
- The number of connections
- Cache hit
- Dead locks
- etc ...

13.18.1 Introduction

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you follow the quickstart.

13.18.2 Overview

Note: TODO: draw a check_mssql diag

Monitoring a Mssql server need the plugin check_mssql_health available at **'labs.consol.de/lang/en/nagios/check_mssql_health/'_** and a mssql user account for the connection.

13.18.3 Steps

There are some steps you'll need to follow in order to monitor a new database machine. They are:

- Install check plugins
- setup the mssql user account
- Update your windows server host definition for mysql monitoring

- Restart the Shinken daemon

13.18.4 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- Some **check_mssql_** commands definition has been added to the "commands.cfg" file.
- A Mssql host template (called "mssql") has already been created in the "templates.cfg" file.

The above-mentioned config files can be found in the ///etc/shinken/// directory (or *c:shinkenetc* under windows). You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your Mssql boxes in no time.

Tip: We are supposing here that the Mssql machine you want to monitor is named srv-win-1. Please change the above lines and commands with the real name of your server of course.

13.18.5 Installing the check plugins on Shinken

First connect as root under you Shinken server (or all poller servers for a multi-box setup)

Note: Todo: Use shinken.sh for this

13.18.6 Setup the mssql user account

Look at the **'labs.consol.de/lang/en/nagios/check_mssql_health/'_** page about how to configure your user connection.

Then you will need to configure your user/password in the macros file so the plugins will have the good values for the connction. So update the /etc/shinken/resource.cfg file or c:shinkenetcresource.cfg file to setup the new password:

```
$MSSQLUSER$=shinken
$MSSQLPASSWORD$=shinkenpassword
```

Test the connection

To see if the connection is ok, just launch:

```
::
```

> /var/lib/nagios/plugins/check_mssql_health –hostname srv-win-1 –username shinken –password shinkenpassword –mode connection-time

It should not return errors.

13.18.7 Declare your host as a mssql server

All you need to get all the Msql service checks is to add the *mssql* template to this host. We suppose you already monitor the OS for this host, and so you already got the host configuration file for it.

Under Linux:

```
: :
```

linux:~ # vi /etc/shinken/hosts/srv-win-1.cfg

Or Windows:

```
: :
```

c:wordpad c:shinkenetchostssrv-win-1.cfg

You need to add the mysql template in the use line. It's better to follow the more precise template to the less one, like here mssql first, and then windows.

```
: :
```

define host{ use mssql,windows host_name srv-win-1 address srv-win-1.mydomain.com

}

What is checked with a mssql template?

At this point, you configure your host to be checked with a mssql template. What does it means? It means that you got some ser

- connection-time: Measures how long it takes to login 0..n seconds (1, 5)
- connected-users: Number of connected users 0..n (50, 80)
- cpu-busy: CPU Busy Time 0%..100% (80, 90)
- io-busy: IO Busy Time 0%..100% (80, 90)
- full-scans: Number of Full Table Scans per second 0..n (100, 500)
- transactions: Number of Transactions per second 0..n (10000, 50000)
- batch-requests: Number of Batch Requests per second 0..n (100, 200)
- latches-waits: Number of Latch-Requests per second, which could not be fulfilled 0..n (10, 50)
- latches-wait-time: Average time a Latch-Request had to wait until it was granted 0..n ms (1, 5)
- locks-waits: Number of Lock-Requests per second, which could not be satisfied. 0..n (100, 500)
- locks-timeouts: Number of Lock-Requests per second, which resulted in a timeout. 0..n (1, 5)
- locks-deadlocks: Number of Deadlocks per second 0..n (1, 5)
- sql-recompilations: Number of Re-Compilations per second 0..n (1, 10)
- sql-initcompilations: Number of Initial Compilations per second 0..n (100, 200)
- total-server-memory: The main memory reserved for the SQL Server 0..n (nearly1G, 1G)
- mem-pool-data-buffer-hit-ratio: Data Buffer Cache Hit Ratio 0%..100% (90, 80:)
- lazy-writes: Number of Lazy Writes per second 0..n (20, 40)
- page-life-expectancy: Average time a page stays in main memory 0..n (300:, 180:)
- free-list-stalls: Number of Free List Stalls per second 0..n (4, 10)
- checkpoint-pages: Number of Flushed Dirty Pages per second 0..n ()
- database-free: Free space in a database (Default is percent, but –units can be used also). You can select a single database with the name parameter. 0%..100% (5%, 2%)

- database-backup-age Elapsed time since a database was last backupped (in hours). The performancedata also cover the time needed for the backup (in minutes). 0..n

13.18.8 Restarting Shinken

You're done with modifying the Shiknen configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors! .. _labs.consol.de/lang/en/nagios/check_mssql_health/: http://labs.consol.de/lang/en/nagios/check_mssql_health/

13.19 Monitoring MySQL

Abstract

This document describes how you can monitor a MySQL server such as:

- Connection time
- A recent restart
- The number of connections
- Cache hit
- etc...

13.19.1 Introduction

These instructions assume that you've installed Shinken according to the *Installation tutorial*. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you follow the quickstart.

13.19.2 Overview

Note: TODO: draw a check_mysql diag

Monitoring a MySQL server need the plugin check_mysql_health available at **'labs.consol.de/lang/en/nagios/check_mysql_health/'_** and a mysql user account for the connection.

13.19.3 Steps

There are some steps you'll need to follow in order to monitor a new Linux machine. They are:

- Install check plugins
- setup the mysql user account
- Update your server host definition for mysql monitoring
- Restart the Shinken daemon

13.19.4 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- Some **check_mysql_** commands definition has been added to the "commands.cfg" file.
- A Mysql host template (called "mysql") has already been created in the "templates.cfg" file.

The above-mentioned config files can be found in the ///etc/shinken/// directory (or *c:shinkenetc* under windows). You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your Mysql boxes in no time.

Tip: We are supposing here that the Mysql machine you want to monitor is named srv-lin-1 and is a Linux. Please change the above lines and commands with the real name of your server of course.

13.19.5 Installing the check plugins on Shinken

First connect as root under you Shinken server (or all poller servers for a multi-box setup) and launch:

```
shinken.sh -p check_mysql_health
```

13.19.6 Setup the mysql user account

Connect with a root account on your MySQL database. change 'password' with your mysql root password:

```
lin-srv-1:# mysql -u root -ppassword
```

And create a shinken user:

```
GRANT usage ON *.* TO 'shinken'@'%' IDENTIFIED BY 'shinkenpassword';
```

It's a good thing to change the shinkenpassword to another password. Then you need to update the /etc/shinken/resource.cfg file or c:shinkenetcresource.cfg file to setup the new password:

```
$MYSQLUSER$=shinken
$MYSQLPASSWORD$=shinkenpassword
```

Test the connection

To see if the connection is okay, just launch:

```
::
```

> /var/lib/nagios/plugins/check_mysql_health –hostname srv-lin-1 –username shinken –password shinkenpassword –mode connection-time

It should not return errors.

13.19.7 Declare your host as a mysql server

All you need to get all the MySQL service checks is to add the *mysql* template to this host. We suppose you already monitor the OS (linux or windows for example) for this host, and so you already got the host configuration file for it.

Under Linux:

```
::
```

linux:~ # vi /etc/shinken/hosts/srv-lin-1.cfg

Or Windows:

```
::
```

c:wordpad c:shinkenetchostssrv-lin-1.cfg

You need to add the mysql template in the use line. It's better to follow the more precise template to the less one, like here mysql first, and then linux.

```
::
```

define host{ use mysql,linux host_name srv-lin-1 address srv-lin-1.mydomain.com

}

What is checked with a mysql template?

At this point, you configure your host to be checked with a mysql template. What does it means? It means that you got some services checks already configured for you. Warning and alert levels are between ():

- connection-time: Determines how long connection establishment and login take, 0..n Seconds (1, 5)
- uptime: Time since start of the database server (recognizes DB-Crash+Restart), 0..n Seconds (10:, 5: Minutes)
- threads-connected: Number of open connections, 1..n (10, 20)
- threadcache-hitrate: Hitrate in the Thread-Cache 0%..100% (90:, 80:)
- querycache-hitrate: Hitrate in the Query Cache 0%..100% (90:, 80:)
- querycache-lowmem-prunes: Displacement out of the Query Cache due to memory shortness n/sec (1, 10)
- keycache-hitrate: Hitrate in the Myisam Key Cache 0%..100% (99:, 95:)
- bufferpool-hitrate: Hitrate in the InnoDB Buffer Pool 0%..100% (99:, 95:)
- bufferpool-wait-free: Rate of the InnoDB Buffer Pool Waits 0..n/sec (1, 10)
- log-waits: Rate of the InnoDB Log Waits 0..n/sec (1, 10)
- tablecache-hitrate: Hitrate in the Table-Cache 0%..100% (99:, 95:)
- table-lock-contention: Rate of failed table locks 0%..100% (1, 2)
- index-usage: Sum of the Index-Utilization (in contrast to Full Table Scans) 0%..100% (90:, 80:)
- tmp-disk-tables: Percent of the temporary tables that were created on the disk instead in memory 0%..100% (25, 50)
- slow-queries: Rate of queries that were detected as "slow" 0..n/sec (0.1, 1)
- long-running-procs: Sum of processes that are runnning longer than 1 minute 0..n (10, 20)
- slave-lag: Delay between Master and Slave 0..n Seconds
- slave-io-running: Checks if the IO-Thread of the Slave-DB is running
- slave-sql-running: Checks if the SQL-Thread of the Slave-DB is running
- open-files: Number of open files (of upper limit) 0%..100% (80, 95)
- cluster-ndb-running: Checks if all cluster nodes are running.

13.19.8 Restarting Shinken

You're done with modifying the Shiknen configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors! .. _labs.consol.de/lang/en/nagios/check_mysql_health/: http://labs.consol.de/lang/en/nagios/check_mysql_health/

13.20 Monitoring Publicly Available Services

Abstract

This document describes how you can monitor publicly available services, applications and protocols. By "public" I mean services that are accessible across the network - either the local network or Internet. Examples of public services include "HTTP", "POP3", "IMAP", "FTP", and "SSH". There are many more public services that you probably use on a daily basis. These services and applications, as well as their underlying protocols, can usually be monitored by Shinken without any special access requirements.

13.20.1 Introduction

Private services, in contrast, cannot be monitored with Shinken without an intermediary agent of some kind. Examples of private services associated with hosts are things like CPU load, memory usage, disk usage, current user count, process information, etc.

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you followed the quickstart.

13.20.2 Plugins For Monitoring Services

When you find yourself needing to monitor a particular application, service, or protocol, chances are good that a *plugin* exists to monitor it. The official Nagios/Shinken plugins distribution comes with plugins that can be used to monitor a variety of services and protocols. There are also a large number of contributed plugins that can be found in the "contrib/" subdirectory of the plugin distribution. The Monitoringexchange.org website hosts a number of additional plugins that have been written by users, so check it out when you have a chance.

If you don't happen to find an appropriate plugin for monitoring what you need, you can always write your own. Plugins are easy to write, so don't let this thought scare you off. Read the documentation on *developing plugins* for more information.

I'll walk you through monitoring some basic services that you'll probably use sooner or later. Each of these services can be monitored using one of the plugins that gets installed as part of the Nagios/Shinken plugins distribution. Let's get started...

13.20.3 The host definition

Before you can monitor a service, you first need to define a host that is associated with the service. If you follow the windows/linux/printer monitoring tutorial, you should be familiar with the process of creating a host and linking your services to it.

For this example, lets say you want to monitor a variety of services on a remote windows host. Let's call that host *srv-win-1*. The host definition can be placed in its own file. Here's what the host definition for *remotehost* might look like if you followed the windows monitoring tutorial:

```
::
```

define host{ use windows host_name srv-win-1 address srv-win-1.mydomain.com

}

13.20.4 Classic services definition with templates

For classic services like HTTP(s) or FTP, there are some ready to run template that you can use. The full service definition is explained in another tutorial Services definitions.

The idea here is just to *tag* your host with what it is providing as network services and automatically get some default checks like Http or Ftp ones.

13.20.5 Monitoring HTTP

Chances are you're going to want to monitor web servers at some point - either yours or someone else's. The **check_http** plugin is designed to do just that. It understands the "HTTP" protocol and can monitor response time, error codes, strings in the returned HTML, server certificates, and much more.

There is already a *http* template for your host that will create for you a Http service. All you need is to add this template on your host, with a comma for separating templates.

So you host definition will now look like:

```
::
```

define host{ use windows,http host_name srv-win-1 address srv-win-1.mydomain.com

}

Note: TODO: write some custom macros for other page check or timeout

It will create a Http service that will look for the "/" page running on *srv-win-1*. It will produce alerts if the web server doesn't respond within 10 seconds or if it returns "HTTP" errors codes (403, 404, etc.). That's all you need for basic monitoring. Pretty simple, huh?

13.20.6 Monitoring HTTPS

We got more an more HTTPS services. You will basically check two things: page accessibility and certificates.

There is already a *https* template for your host that will create for you a Https and a HttpsCertificate services. The Https check is like the Http one, but on the SSL port. The HttpsCertificate will look for the expiration of the certificate, and will warn you 30 days before the end of the certificate, and raise a critical alert if its expired.

So you host definition will now look like:

```
::
```

define host{ use windows,https host_name srv-win-1 address srv-win-1.mydomain.com

}

Note: TODO: write some custom macros for other page check or timeout

You can check Http AND Https in the same time, all you need is to se the two templates in the same time:

```
::

    define host{ use windows,http,https host_name srv-win-1 address srv-win-1.mydomain.com

    }
```

13.20.7 Monitoring FTP

When you need to monitor "FTP" servers, you can use the **check_ftp** plugin. Like for the Http case, there is already a ftp template that you can use.

```
::

    define host{ use windows,ftp host_name srv-win-1 address srv-win-1.mydomain.com

    }
```

This service definition will monitor the "FTP" service and generate alerts if the "FTP" server doesn't respond within 10 seconds.

13.20.8 Monitoring SSH

When you need to monitor "SSH" servers, you can use the **check_ssh** plugin.

```
::

    define host{ use windows,ssh host_name srv-win-1 address srv-win-1.mydomain.com

    }
```

Tip: You don't need to declare the ssh template if you already configure your host with the linux one, the Ssh service is already configured.

This definition will monitor the "Ssh" service and generate alerts if the "SSH" server doesn't respond within 10 seconds.

13.20.9 Monitoring SMTP

The **check_smtp** plugin can be using for monitoring your email servers. You can use the smtp template for you host to automatically get a Smtp service check.

```
::

    define host{ use windows,smtp host_name srv-win-1 address srv-win-1.mydomain.com

    }
```

This service definition will monitor the "Smtp" service and generate alerts if the "SMTP" server doesn't respond within 10 seconds.

13.20.10 Monitoring POP3

The **check_pop** plugin can be using for monitoring the "POP3" service on your email servers. Use the *pop3* template for your host to get automatically a Pop3 service.

```
::
```

define host{ use windows,pop3 host_name srv-win-1 address srv-win-1.mydomain.com

}

This service definition will monitor the "POP3" service and generate alerts if the "POP3" server doesn't respond within 10 seconds.

13.20.11 Monitoring IMAP

The **check_imap** plugin can be using for monitoring "IMAP4" service on your email servers. You can use the *imap* template for your host to get an Imap service check.

```
::
```

define host{ use windows,imap host_name srv-win-1 address srv-win-1.mydomain.com

}

This service definition will monitor the "IMAP4" service and generate alerts if the "IMAP" server doesn't respond within 10 seconds.

To get smtp, pop3 and imap service checks, you can just link all theses templates to your host:

```
::
```

define host{ use windows,smtp,pop3,imap host_name srv-win-1 address srv-win-1.mydomain.com

}

13.20.12 Restarting Shinken

Once you've added the new host templates to your object configuration file(s), you're ready to start monitoring them. To do this, you'll need to *verify your configuration* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.21 Monitoring an Oracle database server

Abstract

This document describes how you can monitor an Oracle database server such as:

- Connection time
- A recent restart
- The number of connections
- Cache hit
- Dead locks

- etc ...

13.21.1 Introduction

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you follow the quickstart.

13.21.2 Overview

Note: TODO: draw a oracle diag

Monitoring an Oracle server need the plugin check_oracle_health available at **'labs.consol.de/lang/en/nagios/check_oracle_health/'_** and an oracle user account for the connection.

13.21.3 Steps

There are some steps you'll need to follow in order to monitor a new database machine. They are:

```
- Install dependencies
```

- Install check plugins
- Setup the oracle user account
- Creating an alias definition for Oracle databases
- Update your server host definition for oracle monitoring
- Restart the Shinken daemon

13.21.4 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- Some **check_oracle_** commands definition has been added to the "commands.cfg" file.
- An Oracle host template (called "oracle") has already been created in the "templates.cfg" file.

The above-mentioned config files can be found in the ///etc/shinken/// directory (or *c:shinkenetc* under windows). You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your Oracle boxes in no time.

Tip: We are supposing here that the Oracle machine you want to monitor is named srv-lin-1 and is a linux. Please change the above lines and commands with the real name of your server of course.

13.21.5 Installing dependencies

Installing SQL*Plus on the Shinken server

Check_oracle_health plugin needs sqlplus oracle client on the Shinken server, you can download packages on the Oracle Technology Network website. You need to have these 3 packages: oracle-instantclient11.2-basic-11.2.0.3.0-

1.x86_64.rpm oracle-instantclient11.2-devel-11.2.0.3.0-1.x86_64.rpm oracle-instantclient11.2-sqlplus-11.2.0.3.0-1.x86_64.rpm

You can install them like this:

```
linux:~ # rpm -Uvh oracle-instantclient11.2-*
```

Then you have to create some symbolic links in order to have commands in the path:

```
linux:~ # ln -s /usr/lib/oracle/11.2/client64/bin/adrci /usr/bin/adrci
linux:~ # ln -s /usr/lib/oracle/11.2/client64/bin/genezi /usr/bin/genezi
linux:~ # ln -s /usr/lib/oracle/11.2/client64/bin/sqlplus /usr/bin/sqlplus
```

Also, you need to export Oracle environment variables in order to install CPAN modules:

```
linux:~ # export ORACLE_HOME=/usr/lib/oracle/11.2/client64
linux:~ # export PATH=$PATH:$ORACLE_HOME/bin
linux:~ # export LD_LIBRARY_PATH=$ORACLE_HOME/lib
```

Installing CPAN modules

```
linux:~ # perl -MCPAN -e shell
cpan[1]> install DBI
cpan[2]> force install DBD::Oracle
```

13.21.6 Installing the check plugins on Shinken

First connect as root under you Shinken server (or all poller servers for a multi-box setup) and launch:

```
shinken.sh -p check_oracle_health
```

13.21.7 Setup the oracle user account

Tip: You will need to configure the user for all your oracle databases.

Connect to your database as sysadmin on the oracle server:

```
srv-lin-1:oracle# sqlplus "/ as sysdba"
```

And then create your shinken account on the database:

```
CREATE USER shinken IDENTIFIED BY shinkenpassword;
GRANT CREATE SESSION TO shinken;
GRANT SELECT any dictionary TO shinken;
GRANT SELECT ON V_$SYSSTAT TO shinken;
GRANT SELECT ON V_$INSTANCE TO shinken;
GRANT SELECT ON V_$LOG TO shinken;
GRANT SELECT ON SYS.DBA_DATA_FILES TO shinken;
GRANT SELECT ON SYS.DBA_FREE_SPACE TO shinken;
```

And for old 8.1.7 database only:

```
--
-- if somebody still uses Oracle 8.1.7...
GRANT SELECT ON sys.dba_tablespaces TO shinken;
```

```
GRANT SELECT ON dba_temp_files TO shinken;
GRANT SELECT ON sys.v_$Temp_extent_pool TO shinken;
GRANT SELECT ON sys.v_$TEMP_SPACE_HEADER  TO shinken;
GRANT SELECT ON sys.v_$session TO shinken;
```

Then you will need to configure your user/password in the macros file so the plugins will have the good values for the connction. So update the ///etc/shinken/resource.cfg* file or *c:shinkenetcresource.cfg// file to setup the new password:

```
$ORACLEUSER$=shinken
$ORACLEPASSWORD$=shinkenpassword
```

13.21.8 Creating an alias definition for Oracle databases

First, you have to create a tnsnames.ora config file on the shinken server that will contain the alias definition for PROD database:

```
linux:~ # mkdir -p /usr/lib/oracle/11.2/client64/network/admin
linux:~ # vim /usr/lib/oracle/11.2/client64/network/admin/tnsnames.ora
PROD =
```

```
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.0.X)(PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = PROD)
  )
)

:wq
```

Note that you have to declare all databases that you want to monitor with Shinken in this file. For example, if you want to monitor ERP and FINANCE databases, your config file will look like this:

```
ERP =
```

```
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.0.X)(PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = ERP)
  )
)

FINANCE =
```

```
(DESCRIPTION =
  (ADDRESS = (PROTOCOL = TCP)(HOST = 192.168.0.X)(PORT = 1521))
  (CONNECT_DATA =
    (SERVER = DEDICATED)
    (SERVICE_NAME = FINANCE)
  )
)
```

Then, you need define an environment variable that will contain the path of this file with also all others variables related to sqlplus:

```
linux:~ # vi /etc/profile.d/oracle.sh

export PATH=$PATH:/usr/lib/oracle/11.2/client64
export LD_LIBRARY_PATH=/usr/lib/oracle/11.2/client64/lib
export ORACLE_HOME=/usr/lib/oracle/11.2/client64
export TNS_ADMIN=$ORACLE_HOME/network/admin

:wq
```

Adjust rights on the oracle client directory:

```
linux:~ # chown -R shinken:shinken /usr/lib/oracle
```

Optionally, we may have to force loading the oracle client lib like this:

```
linux:~ # vi /etc/ld.so.conf.d/oracle.conf
/usr/lib/oracle/11.2/client64/lib
:wq
linux:~ # ldconfig
```

Test the connection

To see if the connection to the database named PROD is ok, just launch:

```
::
```

> /var/lib/nagios/plugins/check_oracle_health –connect "PROD" –hostname srv-lin-1 –username shinken –password shinkenpassword –mode connection-time

It should not return errors.

Edit shinken init script

Now, you have to edit the shinken init script for loading this new environment:

```
linux:~ # vim /etc/init.d/shinken
(...)
NAME="shinken"

AVAIL_MODULES="scheduler poller reactionner broker receiver arbiter skonf"

# Load environment variables
. /etc/profile.d/oracle.sh

## SHINKEN_MODULE_FILE is set by shinken-* if it's one of these that's calling us.
(...)
:wq
```

13.21.9 Declare your host as an oracle server, and declare your databases

All you need to get all the Oracle service checks is to add the *oracle* template to this host and declare all your databases name. We suppose you already monitor the OS for this host, and so you already got the host configuration file for it.

Under Linux:

```
: :
```

linux:~ # vi /etc/shinken/hosts/srv-lin-1.cfg

Or Windows:

```
: :
```

c:wordpad c:shinkenetchostssrv-lin-1.cfg

You need to add the oracle template in the use line. It's better to follow the more precise template to the less one, like here oracle first, and then linux. You also need to declare in the _databases macros all your database names, separated with comas. Here we suppose you got two databases, ERP and FINANCE (don't forget to declare them into the tnsnames.ora config file such as we described it previously):

```
: :
```

define host{ use oracle,linux host_name srv-lin-1 address srv-lin-1.mydomain.com _databases ERP,FINANCE

}

What is checked with a oracle template?

At this point, you configure your host to be checked with a oracle template. What does it means? It means that you got some services checks already configured for you, and one for each databases you declared. Warning and alert levels are between ():

- tnsping: Listener
- connection-time: Determines how long connection establishment and login take 0..n Seconds (1, 5)
- connected-users: The sum of logged in users at the database 0..n (50, 100)
- session-usage: Percentage of max possible sessions 0%..100% (80, 90)
- process-usage: Percentage of max possible processes 0%..100% (80, 90)
- rman-backup-problems: Number of RMAN-errors during the last three days 0..n (1, 2)
- sga-data-buffer-hit-ratio: Hitrate in the Data Buffer Cache 0%..100% (98:, 95:)
- sga-library-cache-gethit-ratio: Hitrate in the Library Cache (Gets) 0%..100% (98:, 95:)
- sga-library-cache-pinhit-ratio: Hitrate in the Library Cache (Pins) 0%..100% (98:, 95:)
- sga-library-cache-reloads: Reload-Rate in the Library Cache n/sec (10,10)
- sga-dictionary-cache-hit-ratio: Hitrate in the Dictionary Cache 0%..100% (95:, 90:)
- sga-latches-hit-ratio: Hitrate of the Latches 0%..100% (98:, 95:)
- sga-shared-pool-reloads: Reload-Rate in the Shared Pool 0%..100% (1, 10)
- sga-shared-pool-free: Free Memory in the Shared Pool 0%..100% (10:, 5:)
- pga-in-memory-sort-ratio: Percentage of sorts in the memory. 0%..100% (99:, 90:)
- invalid-objects: Sum of faulty Objects, Indices, Partitions
- stale-statistics: Sum of objects with obsolete optimizer statistics n (10, 100)
- tablespace-usage: Used diskspace in the tablespace 0%..100% (90, 98)
- tablespace-free: Free diskspace in the tablespace 0%..100% (5:, 2:)

- tablespace-fragmentation: Free Space Fragmentation Index 100..1 (30:, 20:)
- tablespace-io-balanc: IO-Distribution under the datafiles of a tablespace n (1.0, 2.0)
- tablespace-remaining-time: Sum of remaining days until a tablespace is used by 100%. The rate of increase will be calculated with the values from the last 30 days. (With the parameter –lookback different periods can be specified) Days (90:, 30:)
- tablespace-can-allocate-next: Checks if there is enough free tablespace for the next Extent.
- flash-recovery-area-usage: Used diskspace in the flash recovery area 0%..100% (90, 98)
- flash-recovery-area-free: Free diskspace in the flash recovery area 0%..100% (5:, 2:)
- datafile-io-traffic: Sum of IO-Operationes from Datafiles per second n/sec (1000, 5000)
- datafiles-existing: Percentage of max possible datafiles 0%..100% (80, 90)
- soft-parse-ratio: Percentage of soft-parse-ratio 0%..100%
- switch-interval: Interval between RedoLog File Switches 0..n Seconds (600:, 60:)
- retry-ratio: Retry-Rate in the RedoLog Buffer 0%..100% (1, 10)
- redo-io-traffic: Redolog IO in MB/sec n/sec (199,200)
- roll-header-contention: Rollback Segment Header Contention 0%..100% (1, 2)
- roll-block-contention: Rollback Segment Block Contention 0%..100% (1, 2)
- roll-hit-ratio: Rollback Segment gets/waits Ratio 0%..100% (99:, 98:)
- roll-extends: Rollback Segment Extends n, n/sec (1, 100)
- roll-wraps: Rollback Segment Wraps n, n/sec (1, 100)
- seg-top10-logical-reads: Sum of the userprocesses under the top 10 logical reads n (1, 9)
- seg-top10-physical-reads: Sum of the userprocesses under the top 10 physical reads n (1, 9)
- seg-top10-buffer-busy-waits: Sum of the userprocesses under the top 10 buffer busy waits n (1, 9)
- seg-top10-row-lock-waits: Sum of the userprocesses under the top 10 row lock waits n (1, 9)
- event-waits: Waits/sec from system events n/sec (10,100)
- event-waiting: How many percent of the elapsed time has an event spend with waiting 0%..100% (0.1,0.5)
- enqueue-contention: Enqueue wait/request-Ratio 0%..100% (1, 10)
- enqueue-waiting: How many percent of the elapsed time since the last run has an Enqueue spend with waiting 0%..100% (0.00033,0.0033)
- latch-contention: Latch misses/gets-ratio. With –name a Latchname or Latchnumber can be passed over. (See list-latches) 0%..100% (1,2)
- latch-waiting: How many percent of the elapsed time since the last run has a Latch spend with waiting 0%..100% (0.1,1)
- sysstat: Changes/sec for any value from v$sysstat n/sec (10,10)

13.21.10 Restarting Shinken

You're done with modifying the Shiknen configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors! .. _labs.consol.de/lang/en/nagios/check_oracle_health/: http://labs.consol.de/lang/en/nagios/check_oracle_health/

13.22 Monitoring a printer

Abstract

This document describes how you can monitor a printer. Specifically, HP™ printers that have internal/external JetDirect® cards/devices, or other print servers (like the Troy™ PocketPro 100S® or the Netgear™ PS101®) that support the JetDirect protocol.

The **check_hpjd** plugin (which is part of the standard Nagios/Shinken plugins distribution) allows you to monitor the status of JetDirect-capable printers which have "SNMP" enabled. The plugin is capable of detecting the following printer states:

- Paper Jam
- Out of Paper
- Printer Offline
- Intervention Required
- Toner Low
- Insufficient Memory
- Open Door
- Output Tray is Full
- and more...

13.22.1 Introduction

These instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you follow the quickstart.

13.22.2 Overview

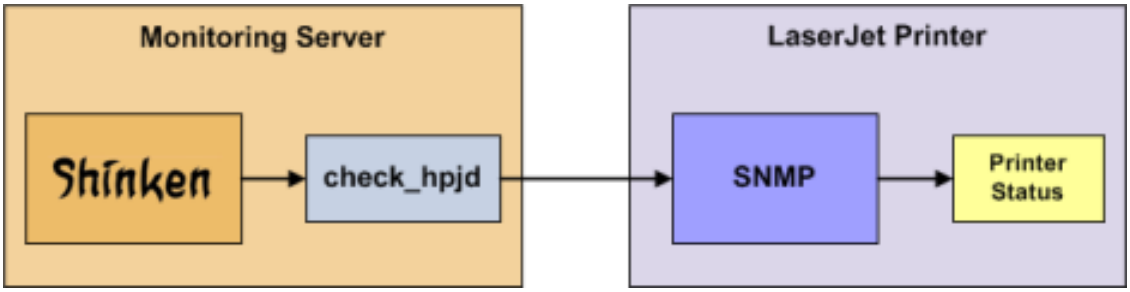

Monitoring the status of a networked printer is pretty simple. JetDirect-enabled printers usually have "SNMP" enabled, which allows Shinken to monitor their status using the **check_hpjd** plugin.

The **check_hpjd** plugin will only get compiled and installed if you have the net-snmp and net-snmp-utils packages installed on your system. Make sure the plugin exists in "/var/lib/nagios/" before you continue. If it doesn't, install net-snmp and net-snmp-utils and recompile/reinstall the Nagios plugins.

13.22.3 Steps

There are some steps you'll need to follow in order to monitor a new printer machine. They are:

- Create new host definition for monitoring this machine
- Restart the Shinken daemon

13.22.4 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A *check_hpjd* command definition has been added to the "commands.cfg" file. This allows you to use the **check_hpjd** plugin to monitor network printers.
- A printer host template (called *printer*) has already been created in the "templates.cfg" file. This allows you to add new printer host definitions in a simple manner.

The above-mentioned config files can be found in the ///etc/shinken/// directory (or *c:shinkenetc* under windows). You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your printers in no time.

Tip: We are supposing here that the printer machine you want to monitor is named printer-1. Please change the above lines and commands with the real name of your printer of course.

13.22.5 Declare your new printer in Shinken

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Linux machine.

You can add the new **host** definition in an existing configuration file, but it's a good idea to have one file by host, it will be easier to manage in the future. So create a file with the name of your server.

Under Linux:

```
: :
```

linux:~ # vi /etc/shinken/hosts/printer-1.cfg

Or Windows:

```
: :
```

c:wordpad c:shinkenetchostsprinter-1.cfg

You need to add a new host definition for the Linux machine that you're going to monitor. Just copy/paste the above definition Change the "host_name", and "address" fields to appropriate values for this machine.

```
: :
```

define host{ use printer host_name printer-1 address 192.160.0.1

}

- The use printer is the "template" line. It mean that this host will **inherits** properties from the printer template. * the host_name is the object name of your host. It must be **unique**. * the address is the network address of your printer. It can be a FQDN or an IP.

What is checked with a printer template?

At this point, you configure your host to be checked with a printer template. What does it means? It means that you got some ch

- printer check each 5 minutes: check with a ping that the printer is UP

Note: TODO: fill what is checked with HPJD

13.22.6 Restarting Shinken

You're done with modifying the Shinken configuration, so you will need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.23 Monitoring Network Devices

Abstract

This document describes how you can monitor network devices (Cisco, Nortel, Procurve,...), such as:

- Network usage
- CPU load
- Memory usage
- Port state
- Hardware state
- etc.

13.23.1 Introduction

These instructions assume that you have installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that are installed if you followed the quickstart.

13.23.2 Overview

Note: TODO: draw a by snmp diag

Network devices are typically monitored using the SNMP and ICMP(ping) protocol.

13.23.3 Steps

Here are the steps you will need to follow in order to monitor a new device:

- Setup check_nwc_health and try a connection with the equipment
- Create new host definition to monitor this device
- Restart the Shinken daemon

13.23.4 What's Already Been Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- A selection of **check_nwc_health** command definitions have been added to the "commands.cfg" file.
- A network equipement host template (called "switch") has already been created in the "templates.cfg" file. This allows you to add new host definitions with a simple keyword.

The above-mentioned configuration files can be found in the ///etc/shinken///packs/network/switch directory (or *c:shinkenetcpacksnetworkswitch* under windows). You can modify the definitions in these and other configuration packs to suit your needs better. However, it is recommended to wait until you are familiar with Shinken before doing so. For the time being, just follow the directions outlined below and you will be securely monitoring your devices in no time.

Tip: In the example, the switch device being monitored is named switch-1. To re-use the example, make sure to update the hostname to that of your device.

13.23.5 Setup check_nwc_health and try a connection switch-1

First connect as the shinken user under your shinken host.

Unix like to install check_nwc_health:

```
install -p check_nwc_health
```

Now to try to check your switch, for this you need a read community for it. Consult your device vendors documentation on how to change the SNMP community. The default value is "public". The most efficient, though less secure protocol version of SNMP is version 2c. Version 3 includes encryption and user/password combinations, but is more convoluted to configure and may tax your devices CPU, it is beyond the scope of this tutorial.

Now connect as the shinken user.

```
su - shinken
```

Warning: NEVER launch plugins like check_* under the root account, because it can work under root but will deposit temporary files that will break the plugins when executed with the shinken user.

Let's say that the switch-1 IP is 192.168.0.1.

```
/usr/local/shinken/libexec/check_nwc_health --hostname 192.168.0.1 --timeout 60 --community "public"
```

It should give you the state of all interfaces.

13.23.6 Declare your switch in Shinken

If the SNMP community value is a global one you are using on all your hosts, you can configure it in the file /etc/shinken/resource.cfg (or c:shinkenresource.cfg under windows) in the line:

```
$SNMPCOMMUNITYREAD$=public
```

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Linux device.

You can add the new **host** definition in an existing configuration file, but it's a good idea to have one file per host, it will be easier to manage in the future. So create a file with the name of your server.

Under Linux:

```
::
```

linux:~ # vi /etc/shinken/hosts/switch-1.cfg

Or Windows:

```
::
```

c:wordpad c:shinkenetchostsswitch-1.cfg

You need to add a new host definition for the switch device that you're going to monitor. Just copy/paste the above definition Change the "host_name", and "address" fields to appropriate values for this device.

```
::
```

define host{ use switch host_name switch-1 address 192.168.0.1

}

- The use switch is the "template" line. It mean that this host will **inherit** properties and checks from the switch template. * the host_name is the object name of your host. It must be **unique**. * the address is the network address or FQDN of your switch.

If you are using a specific SNMP community for this host, you can configure it in the SNMPCOMUNITY host macro like this:

```
::
```

define host{ use switch host_name switch-1 address 192.168.0.1 _SNMPCOMMUNITY password

}

What is checked with a switch template?

At this point, you configure your host to be checked with a switch template. What does it means? It means that you got some che

- host check each 5 minutes: check with a ping that the device is UP
- interface usage
- interface status
- interface errors

For CPU/memory/Hardware checks

Not all devices are managed by check_nwc_health. To know if yours is, just launch:

```
/usr/local/shinken/libexec/check_nwc_health --hostname 192.168.0.1 --timeout 60 --community "public"
```

If it's ok, you can add the "cisco" template for your hosts (even if it's not a cisco device, we are working on getting more templates configuration).

```
::
```

define host{ use cisco,switch host_name switch-1 address 192.168.0.1 _SNMPCOMMUNITY password

}

If it does not work, to learn more about your device, please launch the command:

```
snmpwalk -v2c -c public 192.168.0.1 | bzip2 > /tmp/device.bz2
```

And launch this this command as well:

```
nmap -T4 -O -oX /tmp/device.xml 192.168.0.1
```

Once you have done that, send us the device.bz2 and device.xml files (located in /tmp directory), we will add this new device to the check_nwc_health plugin in addition to the discovery module. With these files please also provide some general information about the device, so we will incorporate it correctly into the discovery module.

13.23.7 Restarting Shinken

You're done with modifying the Shinken configuration, you will need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.24 Monitoring VMware Machines

Abstract

This document describes how you can monitor "private" services and attributes of ESX and VM machines, such as:

- Memory usage
- CPU load

- Disk usage
- Network usage
- etc.

13.24.1 Introduction

In a VMware server we should monitor:

- ESX hosts
- VM started on it

Note: It is a good practice to automatically create dependencies between them so if an ESX goes down, you won't get useless notifications about all VMs on it. Consult the tutorial about the *vmware arbiter module* for more information on this topic..

For theses checks you will need the check_esx3.pl plugin. .. note:: TODO: write in the setup phase about installing it

Note: TODO: draw a by vSphere check

13.24.2 Steps

There are some steps you'll need to follow in order to monitor a new vmware esx/vm machine. They are:

- Create an account on the vsphere console server
- Configure your vsphere server in Shinken
- Create new host definition for monitoring an esx server
- Create new host definition for monitoring a vm machine
- Restart the Shinken daemon

13.24.3 What's already done for you

To make your life a bit easier, a few configuration tasks have already been done for you:

- Some **check_esx_** commands definition has been added to the "commands.cfg" file.
- A VMware ESX host template (called "esx") has already been created in the "templates.cfg" file. This allows you to add new host definitions in a simple manner.
- A VMware virtual machine host template (called "vm") has already been created in the templates.cfg file.

The above-mentioned config files can be found in the ///etc/shinken/// directory (or *c:shinkenetc* under windows). You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your VMware boxes in no time.

Tip: We are supposing here that the esx machine you want to monitor is named srv-esx-1 and the virtual machine is a windows vm named srv-vm-1. Please change the above lines and commands with the real name of your server of course.

13.24.4 Create a vSphere account for Shinken

Please ask your VMware administrator to create a Shinken account (can be a windows domain account) and give it read credential on the vSphere environment (on the root datacenter definition with inheritance).

13.24.5 Setup the vSphere console server connection

You need to configure in the "resource.cfg" file ("/etc/shinken/resource.cfg" under linux or "c:shinkenetcresource.cfg" under wi

- the VSPHERE host connection
- the login for the connection
- the password

```
#### vSphere (ESX) part
$VCENTER$=vcenter.mydomain.com
$VCENTERLOGIN$=someuser
$VCENTERPASSWORD$=somepassowrd
```

You can then try the connection if you already know about an esx host, like for example myesx:

```
::
```

/var/lib/nagios/plugins/check_esx3.pl -D vcenter.mydomain.com -H myesx -u someuser -p somepassword -l cpu

Deploy the public keys on the linux host

When you got the public/private keys, you can deploy the public key to you linux servers.

```
ssh-copy-id  -i /home/shinken/.ssh/id_rsa.pub shinken@srv-lin-1
```

Tip: Change srv-lin-1 with the name of your server.

Test the connection

To see if the keys are working, just launch:

```
::
```

check_by_ssh -H srv-lin-1 -C uptime

It should give you the uptime of the srv-lin-1 machine.

13.24.6 Declare your new host in Shinken

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Linux machine.

You can add the new **host** definition in an existing configuration file, but it's a good idea to have one file by host, it will be easier to manage in the future. So create a file with the name of your server.

Under Linux:

```
: :
```

linux:~ # vi /etc/shinken/hosts/srv-lin-1.cfg

Or Windows:

```
: :
```

c:wordpad c:shinkenetchostssrv-lin-1.cfg

You need to add a new host definition for the Linux machine that you're going to monitor. Just copy/paste the above definition Change the "host_name", and "address" fields to appropriate values for this machine.

```
: :
```

define host{ use esx host_name srv-lin-1 address srv-lin-1.mydomain.com

}

- The use linux is the "template" line. It mean that this host will **inherits** properties from the linux template. * the host_name is the object name of your host. It must be **unique**. * the address is the network address of your linux server.

What is supervised by the linux template?

You have configured your host to the checks defined from the linux template. What does this mean? It means that you have som

- host check each 5 minutes: check with a ping that the server is UP
- check disk space
- check if ntpd is started
- check load average
- check physical memory and swap usage
- check for a recent (less than one hour) reboot

13.24.7 Restarting Shinken

You're done with modifying the configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors!

13.25 Monitoring Windows witn NSClient++

13.25.1 Guideline

Here we will focus here on the windows monitoring with an agent, NSClient++.

It can be get at NSClient++ addon on the Windows machine and using the **check_nt** plugin to communicate with the NSClient++ addon. The **check_nt** plugin should already be installed on the Shinken server if you followed the quickstart guide.

Tip: There are others agent like NC_Net but for the sake of simplicity, we will cover only nsclient++, the most

complete and active one

13.25.2 Steps

There are several steps you'll need to follow in order to monitor a new Windows machine. They are:

- Install a monitoring agent on the Windows machine
- Create new host and add it the nsclient template for monitoring the Windows machine
- Restart the Shinken daemon

13.25.3 What's Already Done For You

To make your life a bit easier, a few configuration tasks have already been done for you:

- Some **check_nt** based commands definition has been added to the "commands.cfg" file. This allows you to use the **check_nt** plugin to monitor Window services.
- A Windows host template (called "windows") has already been created in the "templates.cfg" file. This allows you to add new Windows host definitions in a simple manner.

The above-mentioned config files can be found in the "/etc/shinken/" directory. You can modify the definitions in these and other definitions to suit your needs better if you'd like. However, I'd recommend waiting until you're more familiar with configuring Shinken before doing so. For the time being, just follow the directions outlined below and you'll be monitoring your Windows boxes in no time.

13.25.4 Installing the Windows Agent

Before you can begin monitoring private services and attributes of Windows machines, you'll need to install an agent on those machines. I recommend using the NSClient++ addon, which can be found at http://sourceforge.net/projects/nscplus. These instructions will take you through a basic installation of the NSClient++ addon, as well as the configuration of Shinken for monitoring the Windows machine.

- Download the latest stable version of the NSClient++ addon from http://sourceforge.net/projects/nscplus
- Unzip the NSClient++ files into a new C:NSClient++ directory
- Open a command prompt and change to the C:NSClient++ directory
- Register the NSClient++ system service with the following command (as an administrator):

```
: :
```

cd C:NSClient++ nsclient++ /install

- You can install the NSClient++ systray with the following command ('SysTray' is case-sensitive):

```
: :
```

nsclient++ SysTray

- Open the services manager and make sure the NSClientpp service is allowed to interact with the desktop (see the 'Log On' tab of the services manager). If it isn't already allowed to interact with the desktop, check the box to allow it to.

```
: :
```

C:> nsclient++ /start

- If installed properly, a new icon should appear in your system tray. It will be a yellow circle with a black 'M' inside. - Success! The Windows server can now be added to the Shinken monitoring configuration...

13.25.5 Declare your new host in Shinken

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Windows machine.

We will suppose here that your server is named *srv-win-1*. Of course change this name with the real name of your server.

You can add the new **host** definition in an existing configuration file, but it's a good idea to have one file by host, it will be easier to manage in the future. So create a file with the name of your server.

Under Linux:

```
: :
```

linux:~ # vi /etc/shinken/hosts/srv-win-1.cfg

Or Windows:

```
: :
```

c:wordpad c:shinkenetchostssrv-win-1.cfg

You need to add a new host definition for the Windows machine that you're going to monitor. Just copy/paste the above definition Change the "host_name", and "address" fields to appropriate values for the Windows box.

```
: :
```

define host{ use windows,nsclient++ host_name srv-win-1 address srv-win-1.mydomain.com }

- The use windows and nsclient++ templates in the "use" line. It mean that this host will **inherits** properties from the windows and nsclient++ templates. * the host_name is the object name of your host. It must be **unique**. * the address is ... the network address of your host :)

What is checked with a windows template?

At this point, you configure your host to be checked with a windows template. What does it means? It means that you got some

- host check each 5 minutes: check if the RDP port is open or not.
- check disk spaces
- check if autostarting services are started
- check CPU load
- check memory and swap usage
- check for a recent (less than one hour) reboot

13.26 Monitoring Windows Devices

Abstract

This document describes how you can monitor devices running Microsoft Windows using a predefined template. This template can address:

- Memory usage
- CPU load
- Disk usage
- Service states
- Running processes
- Event logs (Application or system)
- etc.

13.26.1 Introduction

Publicly available services that are provided by Windows machines ("HTTP", "FTP", "POP3", etc.) can be monitored by following the documentation on *Monitoring publicly available services (HTTP, FTP, SSH, etc.)*.

The instructions assume that you've installed Shinken according to the Installation tutorial. The sample configuration entries below reference objects that are defined in the sample config files ("commands.cfg", "templates.cfg", etc.) that were installed.

13.26.2 Overview

Monitoring a windows device is possible using two different methods:

- Agent based: by installing software installed on the server, such as NSClient++
- Agentless: by polling directly Windows via the network using the WMI protocol

This document focuses on the agentless method. The agent based method is described in windows monitoring with nsclient++ <windows monitoring with nsclient++>.

13.26.3 Prerequisites

Have a valid account on the Microsoft Windows device (local or domain account) you will monitor using WMI queries.

13.26.4 Steps

There are several steps you'll need to follow in order to monitor a Microsoft Windows device.

- Install check_wmi_plus plugin on the server running your poller daemons
- Setup an account on the monitored windows server for the WMI queries
- Declare your windows host in the configuration
- Restart the Shinken Arbiter

13.26.5 What's Already Been Done For You

To make your life a bit easier, configuration templates are provided as a starting point:

- A selection of **check_windows** based command definitions have been added to the "commands.cfg" file. This allows you to use the **check_wmi_plus** plugin.

- A Windows host template (called "windows") is included the "templates.cfg" file. This allows you to add new Windows host definitions in a simple manner.

The above-mentioned config files can be found in the ///usr/local/shinken/etc/packs/os/windows// directory. You can modify the definitions in these and other templates to suit your needs. However, wait until you're more familiar with Shinken before doing so. For the time being, just follow the directions outlined below and you will be monitoring your Windows devices in no time.

13.26.6 Setup the check_wmi_plus plugin

The plugin used for windows agent less monitoring is check_wmi_plus. To install it, just launch as root on your shinken server:

```
./install -p check_wmi_plus
```

(install is an executable script in the shinken base directory, previously known as shinken.sh)

13.26.7 Setup a windows account for WMI queries

TODO: write on using less than server admin

You need to configure your user account int the /etc/shinken/resources.cfg file or the c:shinkenetcresource.cfg file under windows with the one you just configured:

```
$DOMAINUSER$=shinken_user
$DOMAINPASSWORD$=superpassword
```

Tip: You can also consult the Nagios documentation which provides a very helpful write up on setting up the domain account and assigning permissions. **'Monitoring_Windows_Using_WMI.pdf'_**

13.26.8 Declare your host in Shinken

Now it's time to define some *object definitions* in your Shinken configuration files in order to monitor the new Windows device.

We will assume that your server is named *srv-win-1*. Replace this with the real hostname of your server.

You can add the new **host** definition in an existing configuration file, but it is good practice to have one file per host, it will be easier to manage in the future. So create a file with the name of your server.

Under Linux:

```
::
```

linux:~ # vi /etc/shinken/hosts/srv-win-1.cfg

Or Windows:

```
::
```

c:wordpad c:shinkenetchostssrv-win-1.cfg

You need to add a new host definition for the Windows device that you will monitor. Just copy/paste the above definition, change the "host_name", and "address" fields to appropriate values.

```
::
```

define host{ use windows host_name srv-win-1 address srv-win-1.mydomain.com }

- use windows is the "template" line. This host will **inherit** properties from the "windows" template. * host_name is the object name of your host. It must be **unique**. * address is the ip address or hostname of your host (FQDN or just the host portion).

Note: If you use a hostname be aware that you will have a DNS dependancy in your monitoring system. Either have a periodically updated local hosts file with all relevant entries, long name resolution caching on your host or use an IP address.

What is monitored by the windows template?

You have configured your host to be checked by the windows template. What does it means? It means that you Shinken will mo

- host check each 5 minutes with a ping
- check disk space
- check if autostarting services are started
- check CPU load (total and each CPU)
- check memory and swap usage
- check for a recent (less than one hour) reboot
- critical/warnings errors in the application and system event logs
- too many inactive RDP sessions
- processes hogging the CPU

13.26.9 Restarting Shinken

You're done with modifying the Shinken configuration, so you'll need to *verify your configuration files* and *restart Shinken*.

If the verification process produces any errors messages, fix your configuration file before continuing. Make sure that you don't (re)start Shinken until the verification process completes without any errors! .. _Monitoring_Windows_Using_WMI.pdf: http://assets.nagios.com/downloads/nagiosxi/docs/Monitoring_Windows_Using_WMI.pdf

CHAPTER 14

How to contribute

14.1 Shinken packs

14.1.1 What are Packs?

Packs are a small subset of configuration files, templates or pictures about a subject, like Linux or windows. It's designed to be a "pack" about how to monitor a specific subject. Everyone can contribute by creating and sending their how packs to the Shinken community website.

Technically, it's a zip file with configuration files and some pictures or templates in it.

Let take a simple example with a linux pack, with only a CPU and a memory services checks. Files between [] are optional.

- templates.cfg -> define the host template for the "linux" tag
- commands.cfg -> define commands for getting CPU and Memory information, like by snmp for example.
- linux.pack -> json file that describe your pack
- [discovery.cfg] -> if you got a discovery rule for a "linux" (like a nmap based one), you can add it
- services/
- services/cpu.cfg -> Your two services. They must apply to the "linux" host tag!
- services/memory.cfg
- [images/sets/tag.png] -> if you want to put a linux logo for all linux host tagged host, it's here. (size = 40x40px, in png format)
- [templates/pnp/check_cpu.php] -> if your check_cpu command got a PNP4 template
- [templates/graphite/check_cpu.graph] -> same, but for graphite

14.1.2 What does a .pack file looks like ?

It's a json file that describe your "pack". It will give it its name, where it should be installed, and if need give some macros provided by the host tags so it will be easier to configure in the SkonfUI.

Let use our previous Linux sample:

```
{
"name":"linux",
"description":"Standard linux checks, like CPU, RAM and disk space. Checks are done by SNMP.",
"path":"os/",
"macros":{
```

```
        "_SNMPCOMMUNITY": {"type":"string",
                           "description":"The read snmp community allowed on the linux server"
                           },

        "_LOAD_WARN": {"type":"string",
                           "description": "Value for starting warning state for the load average at 1
                           },
        "_LOAD_CRIT": {"type":"string",
                           "description": "Value for starting critical state for the load average at
                           },

        "_MEM_WARN": {"type":"percent",
                              "description": "Warning level for used disk space"
```

```
                            },
        "_MEM_CRIT": {"type":"percent",
                               "description": "Critical level for used disk space"
                            },

        "_CPU_WARN": {"type":"percent",
                               "description": "Warning level for the CPU usage"
                            }
       }
}
```

Name and description are quite easy to understand. Path is where this pack will be installed on the etc/packs directory for the user (and where it will be push in the community website). It can be a full path, like os/unix/ if you want.

Macros is a hash map of macro names with two values: type and description. Type can be in "string" or "percent" from now. It will give a simple input in SkonfUI for the SNMPCOMMUNITY macro for example, but a "slider" for warning/critical values (so a value between 0 and 100, integer).

14.1.3 How to create a zip pack from a configuration directory?

If you push a valid .pack in the configuration directory in etc/packs, you will be able to create a zip pack with the shinken-packs command. Let use again our linux sample. For example, the configuration directory with templates.cfg, commands.cfg, discovery.cfg, the linux.pack file and the services directory is in etc/packs/os/linux.

```
shinken-packs -c -s /var/lib/shinken/share -p etc/packs/os/linux/linux.pack
```

It will generate the file /tmp/linux.zip for you. All you need to give to the shinken-packs command is the .pack path, and in option the share path where you have your images and templates directory. The command will automatically parse this directory and take the good image set by looking a the pack name (like configured in the .pack file) and took each template that is available for your commands in the commands.cfg file.

You are done. Your zip pack is done and available in /tmp/linux.zip Congrats! :)

14.1.4 How to share the zip pack to the community website?

The community website is available at community.shinken-monitoring.org. You will need an account to share your zip packs or retrive some new from others community members.

Register on the community website and retrieve your API key

You can register online at community.shinken-monitoring.org/register or you can also do it in a CLI way with your shinken-packs command.

Tip: You only need to register once. All your packs will be pushed with the same account.

Note: You will also need the python-pycurl library, but it should be ok on most distribution already.

You need a valid email for registering (so you can validate your account).

```
bin/shinken-packs -r -l mylogin -P mypassword -e email@google.com
```

Tip: If you are behind a proxy, you will need to add a –proxy http://user:password@proxy-server:3128 argument in your shinken-packs command. It will be need for all community calls like registering or pushing zip packs.

You will have an email with a link to validate your email (so we will only spam users that want it :))

In order to push or retrieve packs you will need an api_key that will be generated when you will validate your account. You can connect to the community website and go in your account panel to get it, or you can get it from the shinken-packs command.

```
bin/shinken-packs -g --login mylogin -P mypassword
```

It will give you your api_key, something that looks like d9be716aad1d41988ad87b1a454274a50.

Push your zip pack

Now you got your can push your /tmp/linux.zip pack and make it available for the community!

```
bin/shinken-packs -u -k d9be716aad1d41988ad87b1a454274a50 -z /tmp/linux.zip
```

Then it's done! You can go to the community website and look at your new shared zip pack. Thanks a lot for sharing :)

14.2 Shinken modules and Shinken packs

14.2.1 Packages layout

For a MODULE named ABC (ex: github.com/shinken-monitoring/mod-ip-tag)

- etc/modules/abc.cfg
- module/module.py
- /__init__.py
- package.json

That"s ALL!

For a PACK named ABC (for an example look at github.com/shinken-monitoring/pack-linux-snmp

```
pack/templates.cfg
     /services/
     /commands.cfg
     [.......]
share/images/sets/ABC/tag.png  (if need)
     /templates/graphite/mycommand.graph
```

14.2.2 The package.json file

The package.json is like this for a PACK:

```
{
```

```
 "name": "linux-snmp",
 "types": ["pack"],
 "version": "1.4",
 "homepage": "https://github.com/shinken-monitoring/pack-linux-snmp",
 "author": "Jean Gabès",
```

```
 "description": "Linux checks based on SNMP",
 "contributors": [
    {
       "name": "Jean Gabès",
       "email": "naparuba@gmail.com"
    }
 ],
 "repository": "https://github.com/shinken-monitoring/pack-linux-snmp",
 "keywords": [
    "pack",
    "linux",
    "snmp"
 ],
 "dependencies": {
    "shinken": ">=1.4"
 },
 "license": "AGPL"
}
```

And for a module :

```
{
```

```
"name": "ip-tag",
"types": ["module"],
"version": "1.4",
"homepage": "http://github.com/shinken-monitoring/mod-ip-tag",
"author": "Jean Gabès",
"description": "Tag host by their IP ranges",
"contributors": [
  {
    "name": "Jean Gabès",
    "email": "naparuba@gmail.com"
  }
],
"repository": "https://github.com/shinken-monitoring/mod-ip-tag",
"keywords": [
  "module",
  "arbiter",
  "ip"
],
"dependencies": {
  "shinken": ">=1.4"
},
"license": "AGPL"
}
```

14.2.3 How to publish it

Before publishing, you must register an account on shinken.io. Then on your account page on shinken.io/~ you will got your **api_key**. Put it on your ~/.shinken.ini.

Then you can :

```
cd  my-package
shinken publish
```

That's all :)

Note: currently the integration process is a script on the shinken.io website, so you need to ask naparuba@gmail.com to launch it before he take time to put it on a cron :)).

~~NOTOC~~

14.3 Help the Shinken project

How can I help?

This page has moved to : *How to contribute*

14.4 Getting Help and Ways to Contribute

14.4.1 Shinken resources for users (help) :

- Shinken documentation wiki [Jump back to start page]
- Shinken Troubleshooting FAQ
- ' Support Forums'_
- ' Shinken issues and bug reports'_
- ' Shinken Ideas'_

The documentation wiki contains a **Getting started** section for how-to and tutorial related information. It also hosts an official documentation that has the full in-depth details and a how to contribute section where you can learn how to help grow Shinken.

Your input and support is a precious resource. If the documentation is not clear, or you need help finding that nugget of information, the support forum has the answer to your burning questions. The Shinken Ideas page is a good place to let the development team how Shinken can improve to meet new challenges or better serve its user base.

14.4.2 Ways to contribute :

- help on the documentation [This wiki]
- help on updating this web site
- help on tracking and fixing bugs, Shinken is on github to make it easy!
- coding new features, modules and test cases
- performance profiling of the various daemons, interfaces and modules
- providing references for large installations
- submitting ideas to Shinken Ideas
- responding to questions on the forums

//Guidelines and resources are described for users in the first section and power users and developers in the second section.//

14.4.3 Shinken Guidelines for developers and power users :

Guidelines that should be followed when contributing to the code

- Guidelines - *Hacking the code* [Examples of Shinken programming]
- Guidelines - How to add a new WebUI page
- Guidelines - *Test driven development* [How to create and run tests]
- Guidelines - *Programming rules* [Style, technical debt, logging]
- Informational - Feature planning process and release cycle

Resources for developers and power users

- Development - Collaborative code repository on ‘ Shinken github‘_
- Development - Bug tracking on ‘ Shinken github‘_
- Development - Automated test and integration on ‘ Shinken Jenkins server‘_ ‘ Shinken Jenkins server‘_
- Development - The forums are also a good medium to discuss issues ‘ Support Forums‘_
- Development - Developer Mailing list - Register or search the shinken-devel Mailing list

For bug hunting and programming, you will need to look at the “How to hacking” tutorial page.

GitHub offers great facilities to fork, test, commit, review and comment anything related to Shinken. You can also follow the projects progress in real time.

There is a development mailing list where you can join us. Come and let us know what you think about Shinken, propose your help or ask for it. :)

Thank you for your help in making this software an open source success.

Happy New Year to all Shinken followers, users and developers. xkilian (If you find this you are lucky or tracking changes. ;-)

14.5 Shinken Package Manager

Important: I don’t now how you get here :) it’s a poc of the design of a shinken pack manager. A pack can be a module, a configuration pack or what ever you want.

A pack can be about :

- configuration
- module

Each pack should have a pack.json file that describe it.

```
{
```

```
"name": "linux",
"version": "1.2",
"description": "Standard linux checks, like CPU, RAM and disk space. Checks are done by SNMP.",
"type": "configuration",
"dependencies": {
   "shinken" : ">1.2"
},
"repository": {
```

```
  "type": "git",
  "url": "git://github.com/naparuba/pack-cfg-linux.git"
},
"keywords": [
  "linux", "snmp"
],
"author": "Jean Gabès <naparuba@gmail.com>",
"license": "Affero GPLv3",
"configuration":{
  "path":"os/",
  "macros":{
       "_SNMPCOMMUNITY": {"type":"string",
                          "description":"The read snmp community allowed on the linux server"
                          },
            }
}
}
```

And for a module one :

```
{
```

```
"name": "logstore_mongodb",
"version": "1.2",
"description": "Log store module for LiveStatus. Will save the logs into Mongodb.",
"type": "module",
"dependencies": {
   "shinken" : ">1.2",
   "livestatus" : ">1.2"
},
"repository": {
  "type": "git",
  "url": "git://github.com/naparuba/pack-module-logstore_mongodb.git"
},
"keywords": [
  "mongodb", "log", "livestatus"
],
"author": "Gerhard Lausser <>",
"license": "Affero GPLv3"
}
```

The spm command should be really simple to use.

```
spm install linux
```

This will download the linux pack and put the good files into the rigth place.

```
spm search linux
```

This will output all the pack with linux in the name or the description.

```
spm create
```

This will create a .tar.gz file with all inside.

```
spm publish
```

This will push the .tar.gz file to the registry.shinken-montioring.org website. Will use the ~/.spm/api.key for credentials.

```
spm adduser
```

This will try to register you to the registry website. If the username you propose is already defined, propose you to login and get your API key.

CHAPTER 15

Development

15.1 Shinken Programming Guidelines

The Shinken project aims to have good code quality. This is to the benefit of all. Easy to understand code, that is efficient and well documented can do wonders to introduce new developers to a great system and keep existing one happy!

During scores of secret meetings at undisclosed locations in the heart of Eurodisney, the following guidelines were documented.

So here's the how to keep the developers happy! **__Follow these guidelines__**

15.1.1 The python style guide

The python style guide provides coding conventions for Python programmers covering topics such as:

- Whitespace
- Comments
- Imports
- Classes
- Naming conventions

Keep it as a handy reference: ' PEP8 - Python Style guide'_

15.1.2 Reference book

The Art of Readable Code, is a great book that provides a fast read and an immense value in improving the readability and maintainability of code.

15.1.3 Pyro remote Object Library

Shinken uses Pyro extensively to create its cloud-like architecture.:ref:*You can learn more about Pyro <//pypi.python.org/pypi/Pyro4>* if you wish to develop new features that need to use inter-process communications.

15.1.4 The python docstring guide

The Python docstring guide provides insight in how to format and declare *docstrings* which are used to document classes and functions.

Read through it to provide better understanding for yourself and others: ' Python docstring guide'_

15.1.5 Logging is your friend

Shinken provides a logger module, that acts as a wrapper to the Python logging facilities. This provides valuable feedback to users, power users and developers.

The logging functions also provide different levels to distinguish the type logging level at which the messages should appear. This is similar to how syslog classifies messages based on severity levels.

Some log messages will not get the level printed, these are related to state data. Logging levels for logs generated at system startup and displayed in STDOUT are set by default to display all messages. This is normal behaviour.

Once logging is initialized buffered messages are logged to the appropriate level defined in the daemon INI files. (ex. reactionnerd.ini, brokerd.ini, etc.)

Some test cases depend on logging output, so change existing logging messages to your hearts content, but validate all changes against the FULL test suite. *Learn more about using the Shinken test suite*.

Debug: This is the most verbose logging level (maximum volume setting). Consider Debug to be out-of-bounds for a production system and used it only for development and testing. I prefer to aim to get my logging levels just right so I have just enough information and endeavor to log this at the Information level or above.

Information: The Information level is typically used to output information that is useful to the running and management of your system. Information would also be the level used to log Entry and Exit points in key areas of your application. However, you may choose to add more entry and exit points at Debug level for more granularity during development and testing.

Warning: Warning is often used for handled 'exceptions' or other important log events. For example, if your application requires a configuration setting but has a default in case the setting is missing, then the Warning level should be used to log the missing configuration setting.

Error: Error is used to log all unhandled exceptions. This is typically logged inside a catch block at the boundary of your application.

Critical: Critical is reserved for special exceptions/conditions where it is imperative that you can quickly pick out these events. I normally wouldn't expect Fatal to be used early in an application's development. It's usually only with experience I can identify situations worthy of the FATAL moniker experience do specific events become worth of promotion to Critical. After all, an error's an error.

log: This level has been deprecated for NON NAGIOS/SHINKEN STATE messages and should be replaced with one of the approved logging levels listed above. The STATE messages are easy recognize as they are ALL CAPS. Do not mess with these unless you know what you are doing.

15.1.6 Technical debt must be paid

Coding in Shinken should be fun and rewarding.

"Technical debt": all little hacks here and there. __There comes a time, technical debt must be **paid__**. We can have new features very quickly, if authors do not have to bypass numerous hack. We must take some time before each release to pay all technical debt we can. __The less we've got, the easier it is to service and extend the **code__**.

It's the same for the core architecture. A solid and stable architecture allows developers to build on it and add value. A good example is being able to add a parameter with only a single line :)

We must be responsible with new features, if it means they can be used to build more innovation in the future without hacks :)

15.1.7 Where does the fun happen

' GitHub offers great facilities to fork, test, commit, review and comment anything related to Shinken'_.

You can also follow the project progress in real time. .. _ GitHub offers great facilities to fork, test, commit, review and comment anything related to Shinken: https://github.com/naparuba/shinken .. _ Python docstring guide: http://www.python.org/dev/peps/pep-0257/ .. _ PEP8 - Python Style guide: http://www.python.org/dev/peps/pep-0008/

15.2 Test Driven Development

15.2.1 Introduction

Test Driven Development

In a strict sense TDD is:

- Create the test case for your new function (valid and invalid input, limits, expected output)
- Run the test, make sure it fails as planned.
- Code your function, run the test against it and eventually have it succeed.
- Once the function works, re-factor the code for efficiency, readability, comments, logging and anything else required.
- Any scope changes should be handled by new functions, by adding new test cases or by modifying the existing test case (dangerous for existing code).
- See the ‘ Wikipedia article about TDD‘_ for more information.

There are different test levels :

- Commit level tests : These should be as quick as possible, so everyone can launch them in a few seconds.
- Integration tests : Integration tests should be more extensive and can have longer execution periods as these are launched less often (before a release, or in an integration server).

TDD in Shinken

We think all functions and features should have a test case. Shinken has a system to run automated tests that pinpoint any issues. This is used prior to commit, at regular intervals and when preparing a release.

Shinken uses Test Driven Development to achieve agility and stability in its software development process. Developers must adhere to the described methods in order to maintain a high quality standard.

We know some functions will be hard to test (databases for example), so let’s do our best, we can’t have 100% coverage. Tests should cover the various input and expected output of a function. Tests can also cover end-to-end cases for performance, stability and features. Test can also be classified in their time to run, quick tests for commit level validation and integration tests that go in-depth.

We are not saying “all patches should be sent with tests”, but new features/functions should have tests. First, the submitter knows what his code is designed to do, second this will save us from having to create one or more test cases later on when something gets broken. Test examples can be found in the /tests/ directory of the repository or your local installation.

15.2.2 Add test to Shinken

I guess you have come here to chew bubblegum and create tests ... and you are all out of bubblegum! Lucky you are, here is some help to create tests!

Create a test

Tests use standard python tools to run (unittest), so that they have to be well formatted. But don’t worry it’s quite simple. All you have to know is something (a class or a method) containing the “test” string will be run. The typical

way to create a new test is to copy paste from a simple one. We suggest the test_bad_contact_call.py for example which is very small.

Here is the file :

```
from shinken_test import *

class TestConfig(ShinkenTest):

    def setUp(self):
        self.setup_with_file('etc/nagios_bad_contact_call.cfg')

    def test_bad_contact_call(self):
        # The service got a unknow contact. It should raise an error
        svc = self.conf.services.find_srv_by_name_and_hostname("test_host_0", "test_ok_0")
        print "Contacts:", svc.contacts
        self.assert_(svc.is_correct() == False)

if __name__ == '__main__':
    unittest.main()
```

Basically what unittest does (and nosetest too) is run every method containing "test" in all class containing "test". Here there is only one test : test_bad_contact_call The setUp function is a special one : it is called before every test method to set up the test. You can also execute a special function after each test by defining a tearDown() method.

So the only thing you have to do is rename the function and write whatever you want to test inside!

This is more or less the only thing you have to do if you want to write a test from scratch. See http://docs.python.org/2/library/unittest.html for more detail about unittest

If you are testing more complex stuff you may use other test as template. For example broker module tests are a lot different and need some scheduling statement to simulate real beahvior.

Executing tests

Important: YOU NEED PYTHON 2.7 TO RUN THE TESTS. The test scripts use new Assert statements only available starting in Python 2.7.

Once you have done or edited the python file you have to run the test by typing :

```
python test_nameofyourtest.py
```

If there is no error then everything is correct. You can try to launch it with nosetests to double check it.

Note: nosetests has a different behavior than unittests, the jenkins integration is using nosetests, that's why it's a good thing to check before comminting.

The automated tests permit developers to concentrate on what they do best, while making sure they don't break anything. That's why the tests are critical to the success of the project. Shinken aims for "baseline zero bug" code and failure by design development :) We all create bugs, it's the coders life after all, lets at least catch them before they create havoc for users! The automated tests handle regression, performance, stability, new features and edge cases.

The test is ready for commit. The last thing to do is to run the test suit to ensure you do not create code regression. This can be a long step if you have made a big change.

Shell test run

There are basically two ways to run the test list. The first one (easiest) is to run the quick_test shell script. This will basically iterate on a bunch of python files and run them

FIXME : update test list into git and edit end to end script

```
./quick_tests.sh
```

Then you can run the end to end one : \ FIXME : explain what the script does

```
./test_end_to_end.sh
```

It only takes a few seconds to run and you know that you did not break anything (or this will indicate you should run the in-depth integration level tests :)).

If you are adhering to TDD this will validate that your function fails by design or that you have successfully built your function

Integration test run

The other way to do it is run the new_runtest script (which is run on the Jenkins ingration server)

Note: It can be difficult to make it work from scratch as the script create and install a python virtual enviromnt. On the distros, pip dependencies may be difficult to met. Don't give up and ask help on the mailing list!

```
./test/jenkins/new_runtest ./test/jenkins/shorttests.txt ./test/moduleslist COVERAGE PYLINT PEP8
```

For short tests, coverage and python checking. Just put NOCOVERAGE or NOPYLINT or NOPEP8 instead to remove one.

This ensure that the Jenkins run won't fail. It's the best way to keep tests fine.

Tests and integration servers

The integration server is at http://shinken-monitoring.de:8080/

It use the following tests:

- **test/jenkins/runtests[.bat]** it takes the arguments: "file with a list of test_-scripts" [NO]COVERAGE

[NO]PYLINT * test/test_end_to_end.sh

Other integration server is at https://test.savoirfairelinux.com/view/Shinken/

This one use the new_runtest script.

Automated test execution

The Hudson automated test jobs are:

- Shinken
- executed after each git commit
- runtests test/jenkins/shorttests.txt NOCOVERAGE NOPYLINT
- the scripts in shorttests.txt take a few minutes to run

- give the developer feedback as fast as possible (**nobody should git-commit without running tests in his private environment first**)
- Shinken-Multiplatform
- runs 4 times per day
- runtests test/jenkins/longtests.txt NOCOVERAGE NOPYLINT
- linux-python-2.4,linux-python-2.6,linux-python-2.7,windows-python-2.7
- executes _all_ **test_**-scripts we have, so it takes a long time
- Shinken-End-to-End
- runs after each successful Shinken-Multiplatform
- executes the test/test_end_to_end.sh script
- try a direct launch, install then launch, and high availability environment launch.
- Shinken-Code-Quality
- runs once a day
- runtests test/jenkins/longtests.txt COVERAGE PYLINT
- collects metrics for coverage and pylint

On the Jenkins one :

- Shinken-Upstream-Commit-Short-Tests
- executed after each git commit
- ./test/jenkins/new_runtests ./test/jenkins/shorttests.txt ./test/moodulelist COVERAGE PYLINT PEP8
- test also module in a basic way.
- the scripts in shorttests.txt take a few minutes to run
- give the developer feedback as fast as possible (**nobody should git-commit without running tests in his private environment first**)
- Shinken-Upstream-Daily-Full-Tests
- executed every 6 hours
- ./test/jenkins/new_runtest ./test/jenkins/all_tests.txt ./test/moduleslist COVERAGE PYLINT PEP8
- the all_test is regenerated everytime (all test_*.py)
- run all test in all module listed
- give a full view of shinken coverage.

15.3 Nagios Plugin API

15.3.1 Other Resources

If you're looking at writing your own plugins for Nagios or Shinken, please make sure to visit these other resources:

- The official Nagios plugin project website
- The official ' Nagios plugin development guidelines'_

15.3.2 Plugin Overview

Scripts and executables must do two things (at a minimum) in order to function as Shinken plugins:

- Exit with one of several possible return values
- Return at least one line of text output to "STDOUT"

The inner workings of your plugin are unimportant to Shinken, interface between them is important. Your plugin could check the status of a TCP port, run a database query, check disk free space, or do whatever else it needs to check something. The details will depend on what needs to be checked - that's up to you.

If you are interested in having a plugin that is performant for use with Shinken, consider making it a Python or python + Ctype plugin that is daemonized by the Shinken poller or receiver daemons. You can look at the existing poller daemons for how to create a module, it is very simple.

15.3.3 Return Code

Shinken determines the status of a host or service by evaluating the return code from plugins. The following tables shows a list of valid return codes, along with their corresponding service or host states.

Plugin Return Code	Service State	Host State
0	OK	UP
1	WARNING	UP or DOWN/UNREACHABLE*
2	CRITICAL	DOWN/UNREACHABLE
3	UNKNOWN	DOWN/UNREACHABLE

If the use_aggressive_host_checking option is enabled, return codes of 1 will result in a host state of DOWN or UNREACHABLE. Otherwise return codes of 1 will result in a host state of UP. The process by which Nagios determines whether or not a host is DOWN or UNREACHABLE is discussed *here*.

15.3.4 Plugin Output Spec

At a minimum, plugins should return at least one of text output. Beginning with Nagios 3, plugins can optionally return multiple lines of output. Plugins may also return optional performance data that can be processed by external applications. The basic format for plugin output is shown below:

TEXT OUTPUT | OPTIONAL PERFDATALONG TEXT LINE 1LONG TEXT LINE 2...LONG TEXT LINE N | PERFDATA LINE 2PERFDATA LINE 3...PERFDATA LINE N

The performance data (shown in orange) is optional. If a plugin returns performance data in its output, it must separate the performance data from the other text output using a pipe (|) symbol. Additional lines of long text output (shown in blue) are also optional.

15.3.5 Plugin Output Examples

Let's see some examples of possible plugin output...

Case 1: One line of output (text only)

Assume we have a plugin that returns one line of output that looks like this:

```
DISK OK - free space: / 3326 MB (56%);
```

If this plugin was used to perform a service check, the entire line of output will be stored in the "$SERVICEOUTPUT$" macro.

Case 2: One line of output (text and perfdata)

A plugin can return optional performance data for use by external applications. To do this, the performance data must be separated from the text output with a pipe (|) symbol like such:

```
DISK OK - free space: / 3326 MB (56%);

"|"

/=2643MB;5948;5958;0;5968
```

If this plugin was used to perform a service check, the"red"portion of output (left of the pipe separator) will be stored in the $SERVICEOUTPUT$ macro and the"orange"portion of output (right of the pipe separator) will be stored in the $SERVICEPERFDATA$ macro.

Case 3: Multiple lines of output (text and perfdata)

A plugin optionally return multiple lines of both text output and perfdata, like such:

```
DISK OK - free space: / 3326 MB (56%);"|"/=2643MB;5948;5958;0;5968/ 15272 MB (77%);/boot 68 MB (69%);
```

If this plugin was used to perform a service check, the red portion of first line of output (left of the pipe separator) will be stored in the $SERVICEOUTPUT$ macro. The orange portions of the first and subsequent lines are concatenated (with spaces) are stored in the $SERVICEPERFDATA$ macro. The blue portions of the 2nd - 5th lines of output will be concatenated (with escaped newlines) and stored in $LONGSERVICEOUTPUT$ the macro.

The final contents of each macro are listed below:

Macro	Value
$SERVICEOUTPUT$	DISK OK - free space: / 3326 MB (56%);
$SERVICEPERFDATA$	/=2643MB;5948;5958;0;5968"/boot=68MB;88;93;0;98"/home=69357MB;253404;253409;0;253414"/var/log=
$LONGSERVICEOUTPUT$	/ 15272 MB (77%);n/boot 68 MB (69%);n/var/log 819 MB (84%);

With regards to multiple lines of output, you have the following options for returning performance data:

- You can choose to return no performance data whatsoever
- You can return performance data on the first line only
- You can return performance data only in subsequent lines (after the first)
- You can return performance data in both the first line and subsequent lines (as shown above)

15.3.6 Plugin Output Length Restrictions

Nagios will only read the first 4 KB of data that a plugin returns. This is done in order to prevent runaway plugins from dumping megs or gigs of data back to Nagios. This 4 KB output limit is fairly easy to change if you need. Simply edit the value of the MAX_PLUGIN_OUTPUT_LENGTH definition in the include/nagios.h.in file of the source code distribution and recompile Nagios. There's nothing else you need to change!

Shinken behaviour is ... TODO fill in the blanks.

15.3.7 Examples

If you're looking for some example plugins to study, I would recommend that you download the official Nagios plugins and look through the code for various C, Perl, and shell script plugins. Information on obtaining the official Nagios

plugins can be found *here*.

Or go to the Shinken Git hub or look in your installation in shinken/modules and look for the NRPE and NSCA modules for inspiration on create a new poller or receiver daemon module.

15.4 Developing Shinken Daemon Modules

How to develop daemon modules...

Coming shortly. Prev | Up

15.5 Hacking the Shinken Code

15.5.1 Development goals

Shinken is an open source program.

Enhancements, optimization, fixes, these are all good reasons to create a patch and send it to the development mailing list. Even if you are not sure about the quality of your patch or your ideas, submit them to the mailing list for review or comment. Having feedback is essential to any project.

This documentation will show how to add code to Shinken. Shinken is written in Python. If you are new to python please consider reading this ‘ introduction / beginners guide‘_.

15.5.2 Development rules

If you wish to commit code to the Shinken repository, you must strive to follow the Shinken development rules.

15.5.3 How is Shinken’s code organized

The Shinken code is in the shinken directory. All important source files are :

- bin/shinken-* : source files of daemons
- shinken/objects/item.py : base class for nearly all important objects like hosts, services and contacts.
- shinken/*link.py : class used by Arbiter to connect to daemons.
- shinken/modules/*/*py : modules for daemons (like the Broker).
- shinken/objects/schedulingitem.py : base class for host/service that defines all the algorithms for scheduling.

15.5.4 Datadriven programming in Shinken code

A very important thing in Shinken code is the data programming method : instead of hardcoding transformation for properties, it’s better to have a array (dict in Python) that described all transformations we can use on these properties.

With this method, a developer only needs to add this description, and all transformations will be automatic (like configuration parsing, inheritance application, etc).

Nearly all important classes have such an array. It’s named “properties” and is attached to the class, not the object itself.

Global parameters of the application (like the one for nagios.cfg file) are in the properties of the Config class. They are defined like: 'enable_notifications' : {'required':False, 'default':'1', 'pythonize': to_bool, 'class_inherit' : [(Host, None), (Service, None), (Contact, None)]},

Here, this property is:

- not required
- It's default value is 1
- We use the 'to_bool' function to transform the string from the configuration file to a Python object
- We put this value in the Host and Service class with the same name (None==keep the name). The string in place of None, this string is used to access this property from the class.

Specific properties for objects like Hosts or Services are in a properties dict(dictionary), but without the 'class_inherit' part. Instead of this, they have the "fill_brok" part. A "brok" is an inter process message. It's used to know if the property must be sent in the following brok types:

- full_status : send a full status brok, like at daemon starting.
- check_result : send when a check came back
- next_schedule : send when a new check is scheduled

These classes also have another "properties" like dict : "running_properties". It's like the standard one, but for running only properties (aka no configuration based properties).

15.5.5 Programming with broker modules in Shinken

Modules are pieces of code that are executed by a daemon.

Module configuration and startup is controlled in the shinken-specific.cfg

- The module is declared in a daemon
- The module itself is defined and its variables set

A shinken module class must have an _init_, init and documentation. A shinken module can use the following functions:

- managed_broks: A specific function that will dynamically build calls for functions for specific brok.types if the functions exist.
- manage_NAME-OF-BROK-TYPE_broks: The function that will process a specific type of brok

The brok types are created in the code and are not registered in a central repository. At this time the following brok types exist and can be processed by broker modules.

- clean_all_my_instance_id
- host_check_resulthost_next_schedule
- initial
- initial_command_status
- initial_contactgroup_status
- initial_contact_status
- initial_hostgroup_status
- initial_host_status
- initial_poller_status

- initial_reactionner_status
- initial_receiver_status
- initial_scheduler_status
- initial_servicegroup_status
- initial_service_status
- initial_timeperiod_status
- log
- notification_raise
- program_status
- service_check_result
- service_check_resultup
- service_next_schedule
- update
- update_host_status
- update_poller_status
- update_program_status
- update_reactionner_status
- update_receiver_status
- update_scheduler_status
- update_service_status

15.5.6 Example of code hacking : add a parameter for the flapping history

- Configuration part
- Running part
- The perfect patch

In the Nagios code, the flapping state history size is hard coded (20). As in the first Shinken release. Let'S see how it works to add such a parameter in the global file and use it in the scheduling part of the code.

We will see that adding such a parameter is very (very) easy. To do this, only 5 lines need to be changed in :

- config.py : manage the global configuration
- schedulingitem.py : manage the scheduling algorithms of host/services

Configuration part

In the first one (config.py) we add an entry to the properties dict : "flap_history" : {"required":False, "default":'20", "pythonize": to_int, "class_inherit" : [(Host, None), (Service, None)]}, So this property will be an option, with 20 by default, and will be put in the Host and Service class with the name 'flap_history'.

That's all for the configuration! Yes, no more add. Just one line :)

Running part

Now the scheduling part (schedulingitem.py). The hard code 20 was used in 2 functions : add_flapping_change and update_flapping. From this file, we are in an object named self in Python. To access the 'flap_history' of the Host or Service class of this object, we just need to do : flap_history = self.__class__.flap_history Then we change occurrences in the code : if len(self.flapping_changes) > flap_history: [...] r += i*(1.2-0.8)/flap_history + 0.8 r = r / flap_history

That's all. You can test and propose the patch in the devel list. We will thank you and after some patch proposals, you can ask for a git access, you will be a Shinken developer :)

The perfect patch

If you can also add this property in the documentation (/doc directory)

If you followed the Python style guide. (See development rules)

If you created an automated test case for a new feature. (See development rules)

If you documented any new feature in the documentation wiki.

The patch will be __***perfect***__ :)

CHAPTER 16

Shinken modules

This part must be splitted and sent in all module repositories

16.1 Shinken modules

This part must be splitted and sent in all module repositories

16.1.1 Shinken modules

This part must be splitted and sent in all module repositories

Shinken modules

This part must be splitted and sent in all module repositories

Shinken modules

This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Shinken modules This part must be splitted and sent in all module repositories

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
```

```
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP        8.8.4.4
    _EC2_RAMDISKID        None
    _EC2_STATUS           stopped
    _EC2_TAGS             demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
```

```
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    RedisRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus

- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
     broker_name      broker-1
[...]
```

```
     modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
```

```
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis
- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ''service shinken stop''
- Save the pickle: ''pickle.dump(r, open('/tmp/retention.dat', 'w'))''
- Start Shinken: ''service shinken start''

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That's it. From now on all checks that use this command will use Shinken's NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of "Modules" and "Addons" that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken
- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)

- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)
- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
     module_name      Service-Perfdata
     module_type      service_perfdata
     path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
     module_name      Host-Perfdata
     module_type      host_perfdata
     path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
     module_name              Status-Dat
     module_type              status_dat
     status_file              /usr/local/shinken/var/status.data
     object_cache_file        /usr/local/shinken/var/objects.cache
     status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

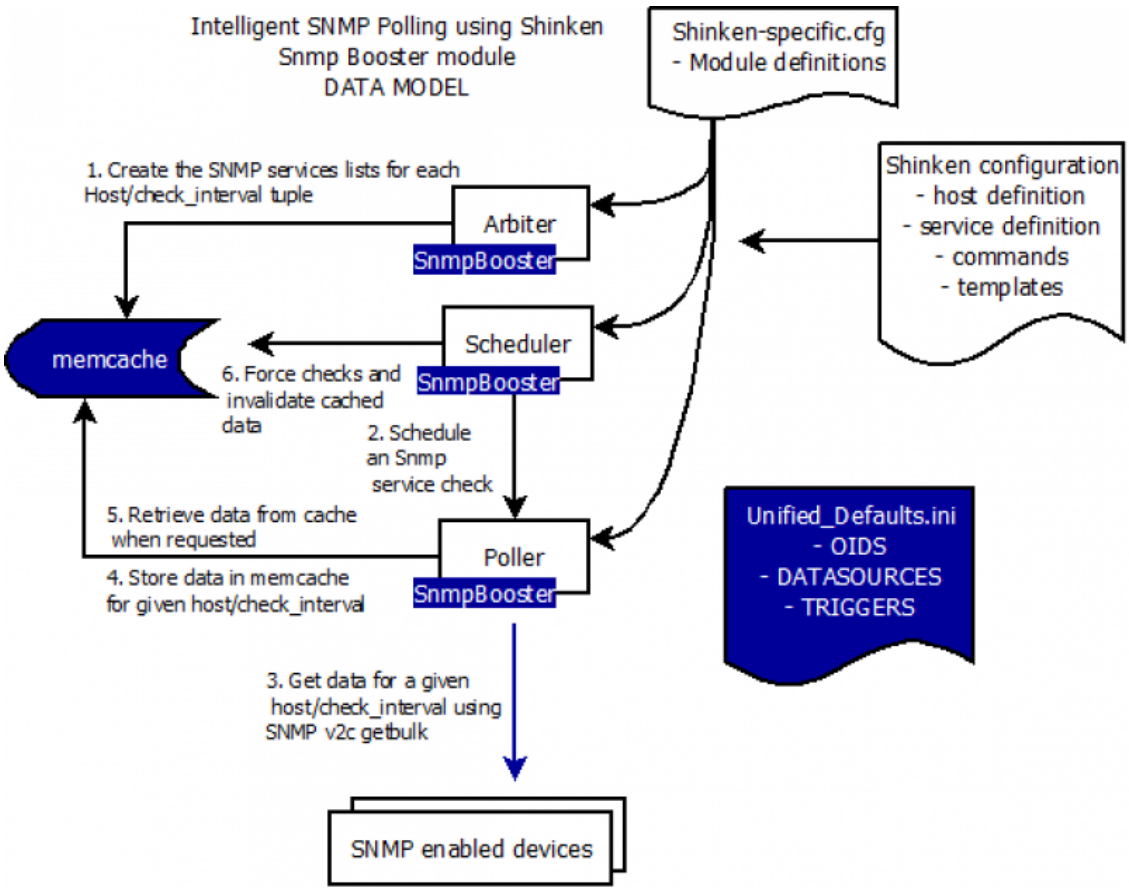

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

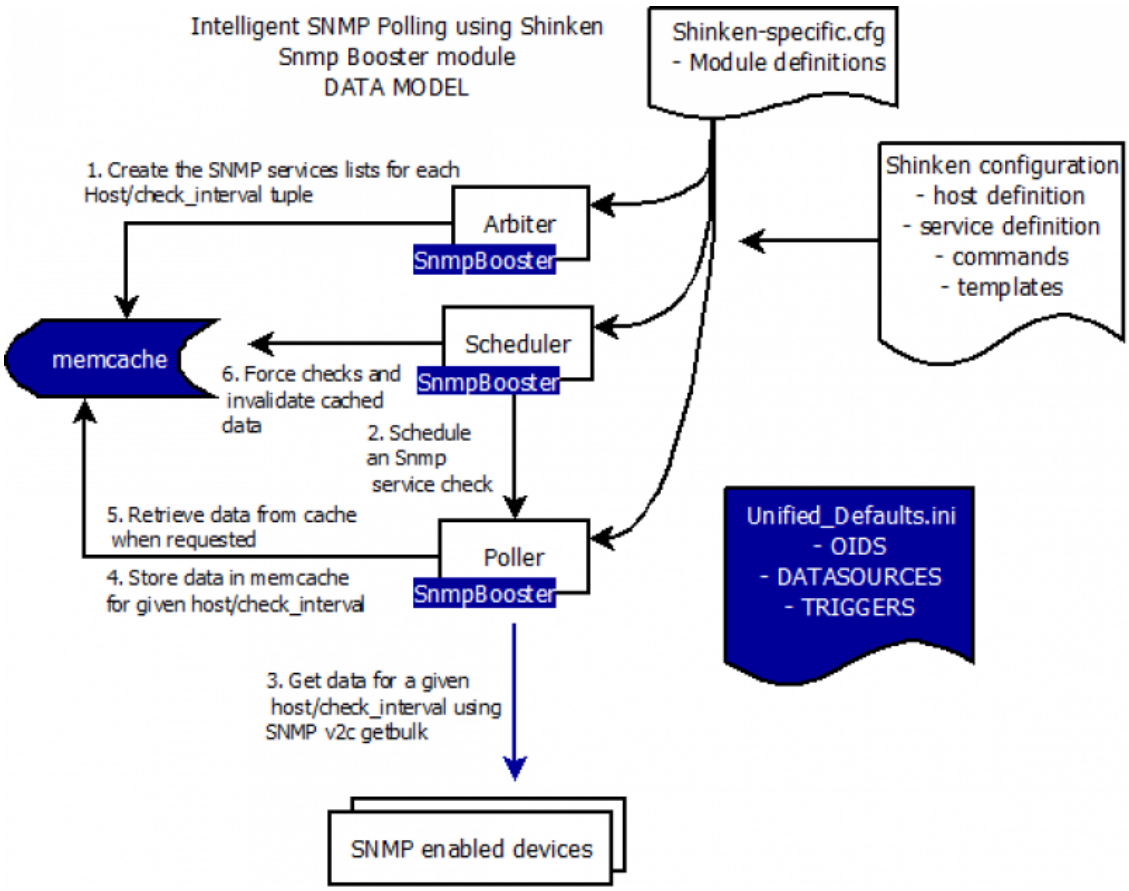

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

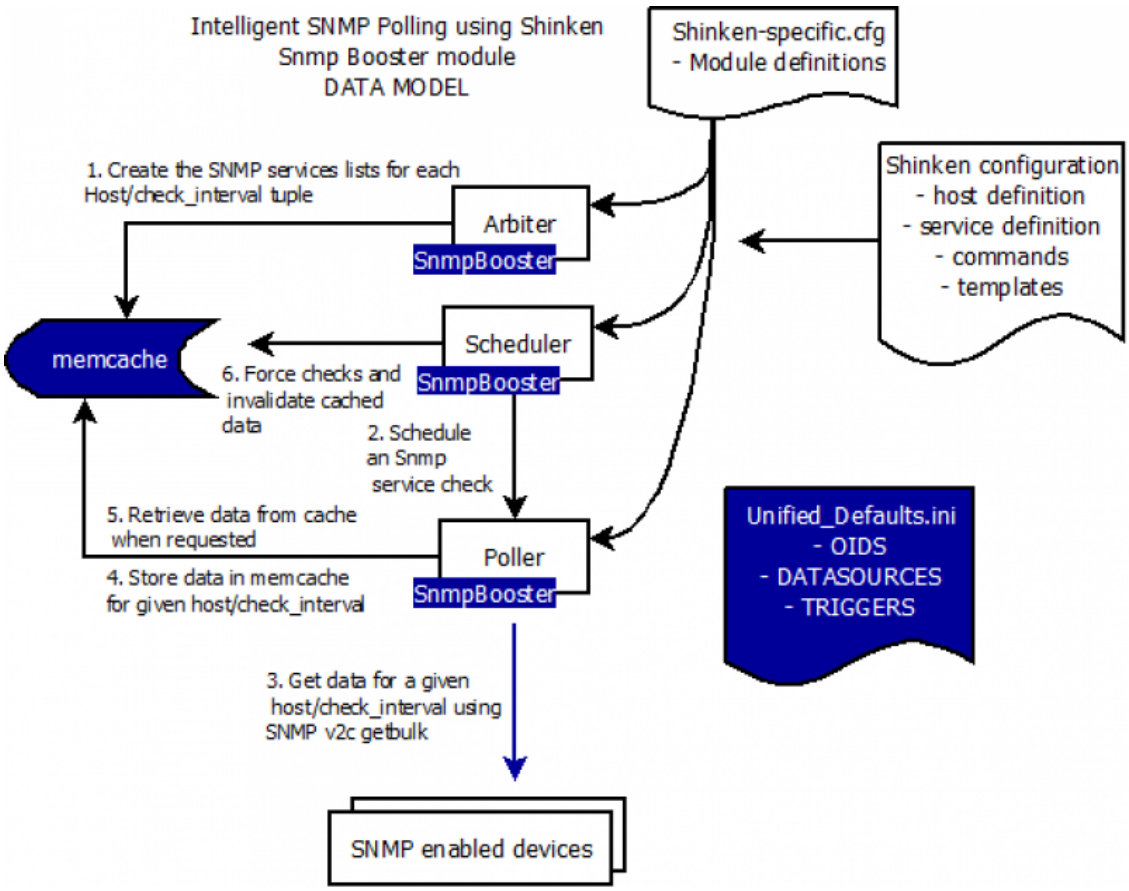

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

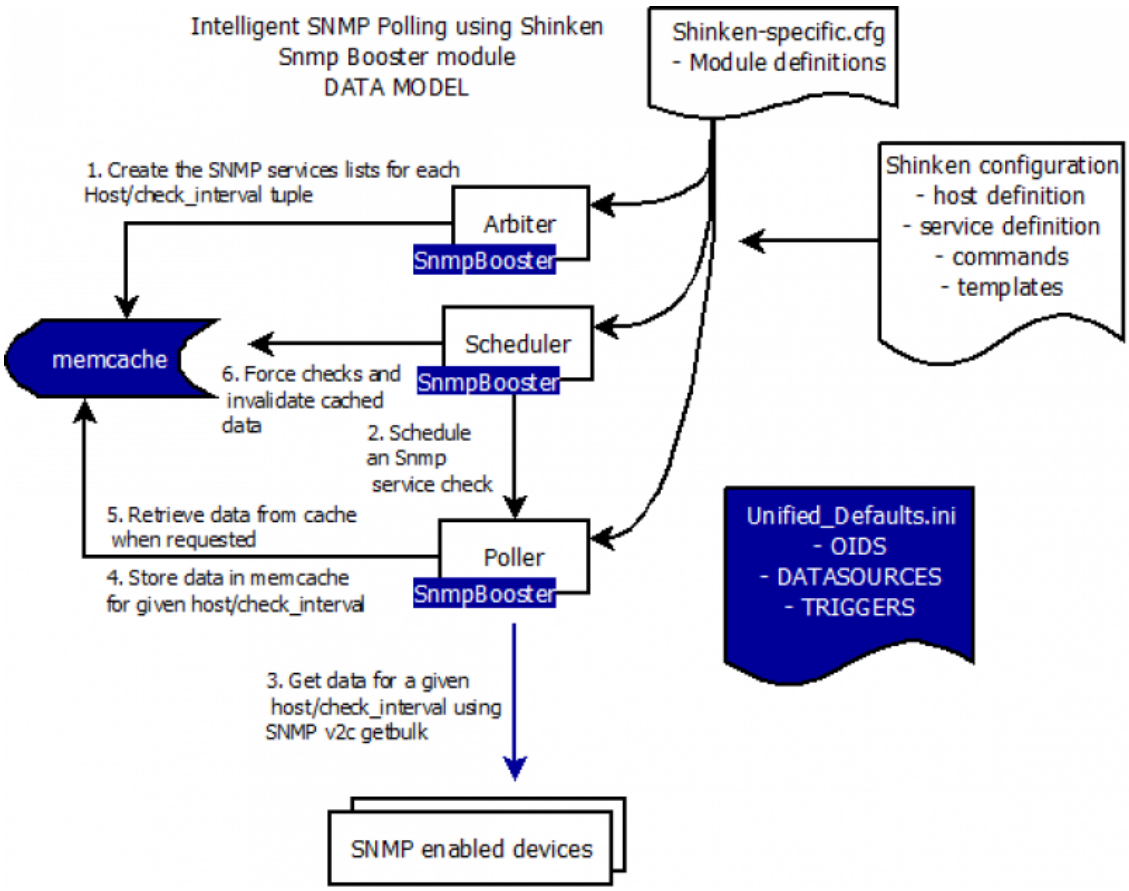

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

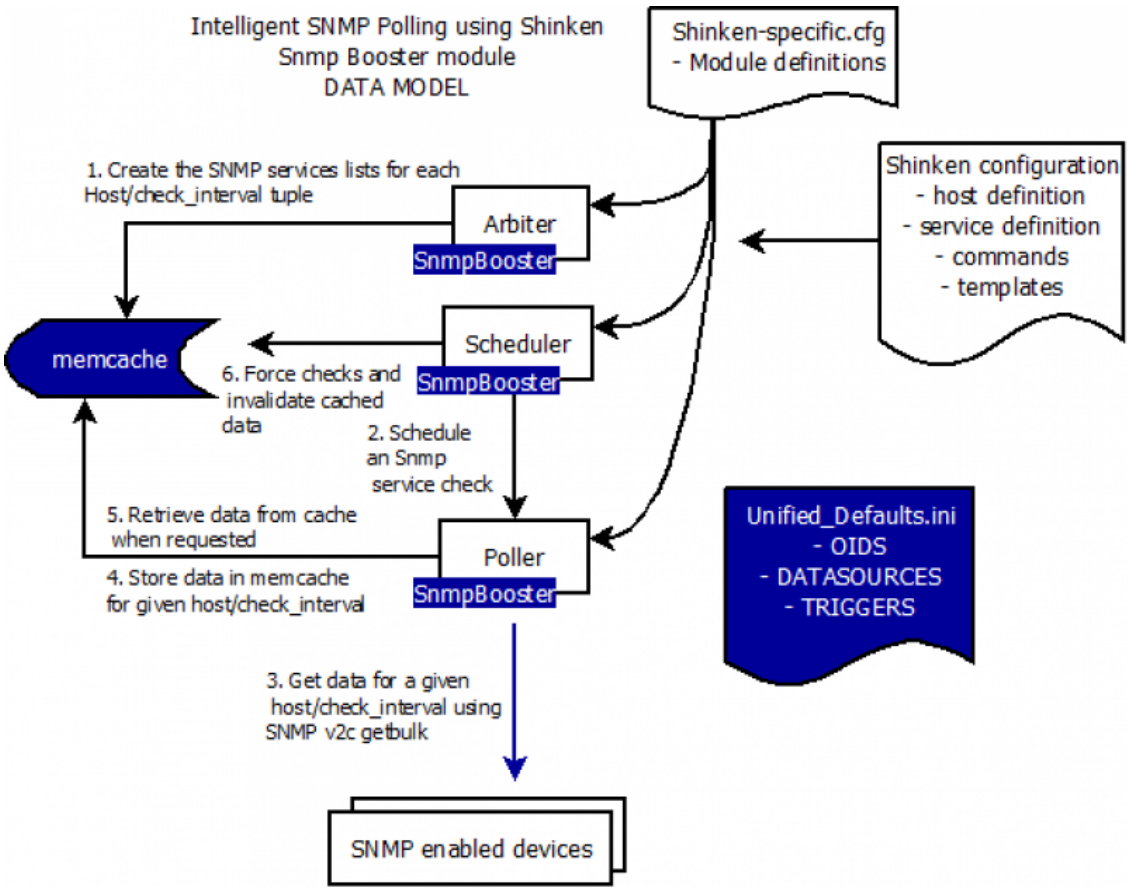

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

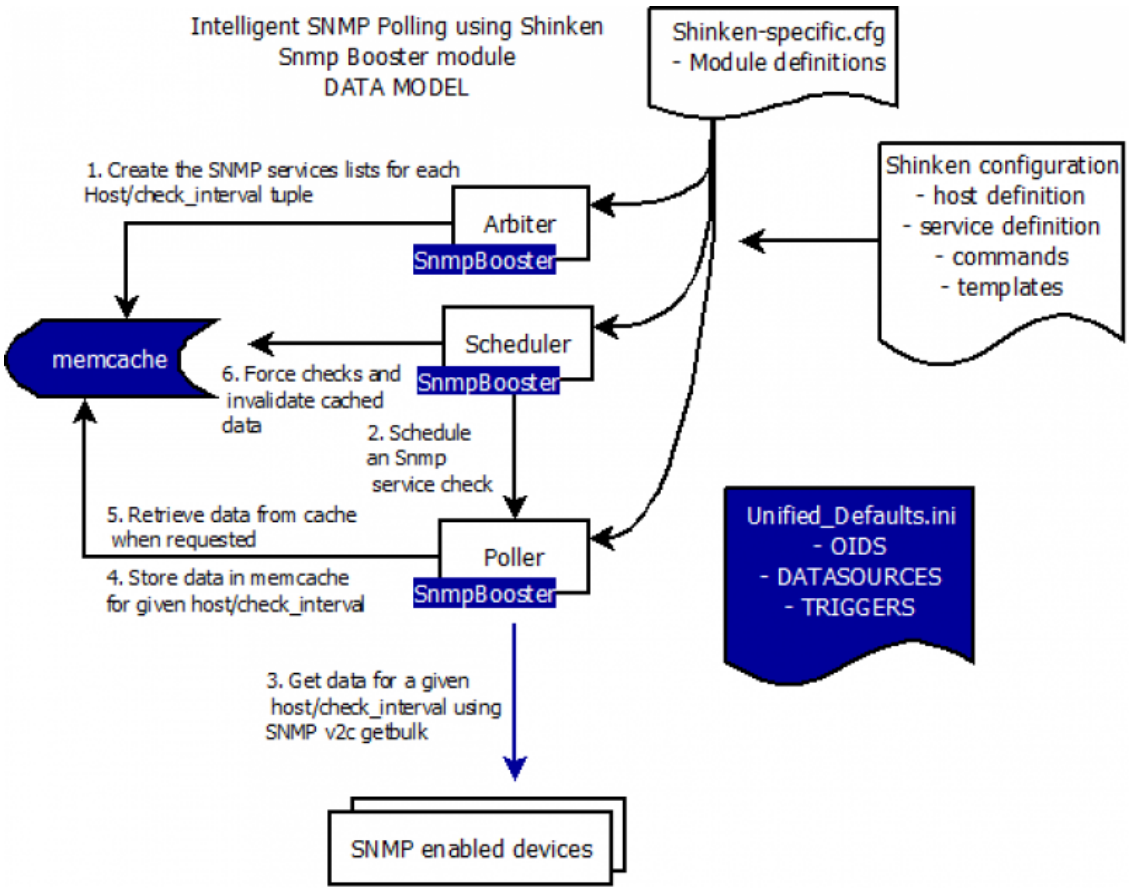

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

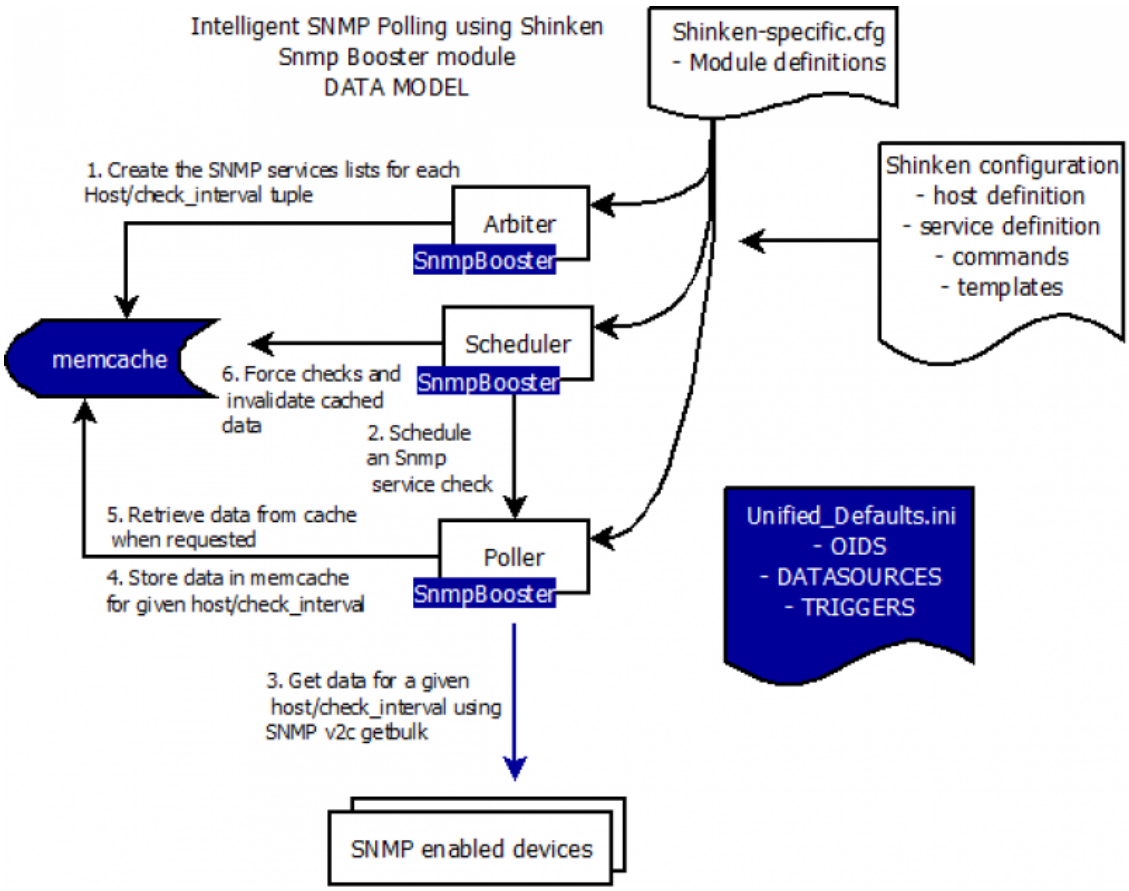

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

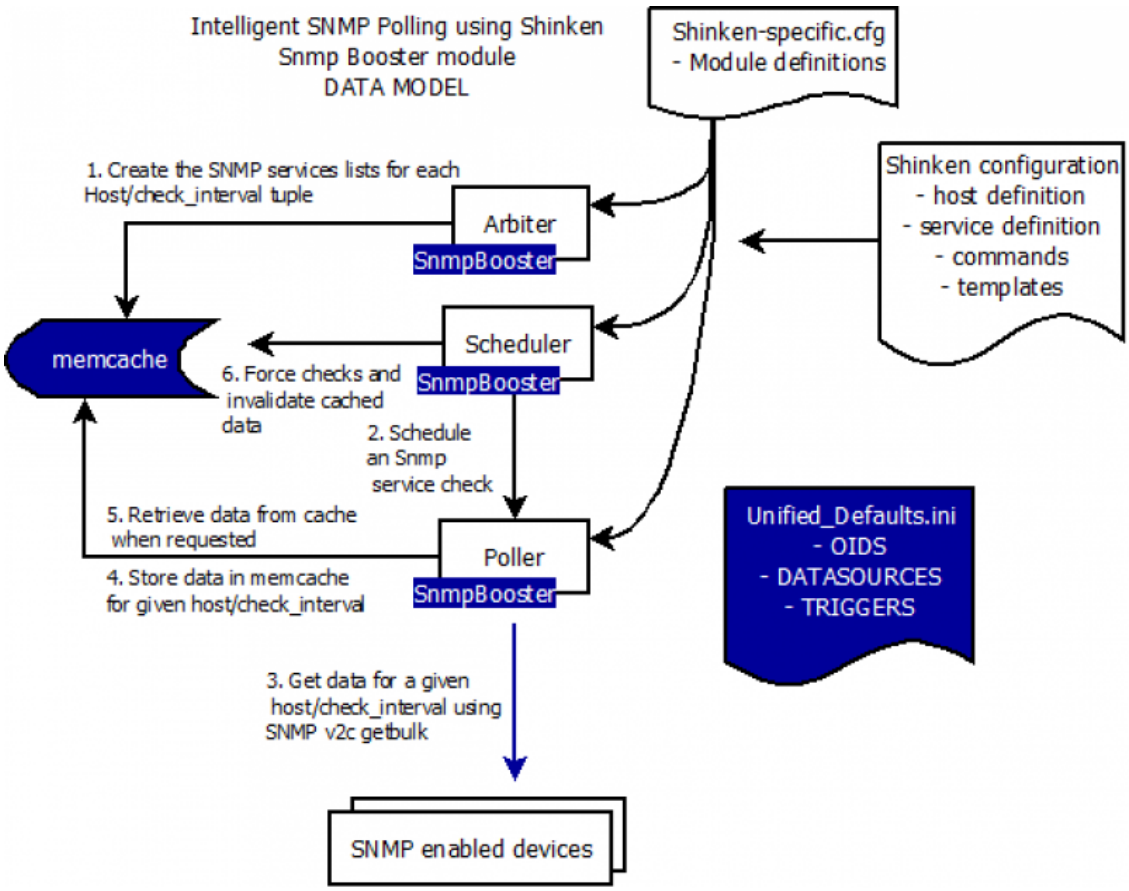

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

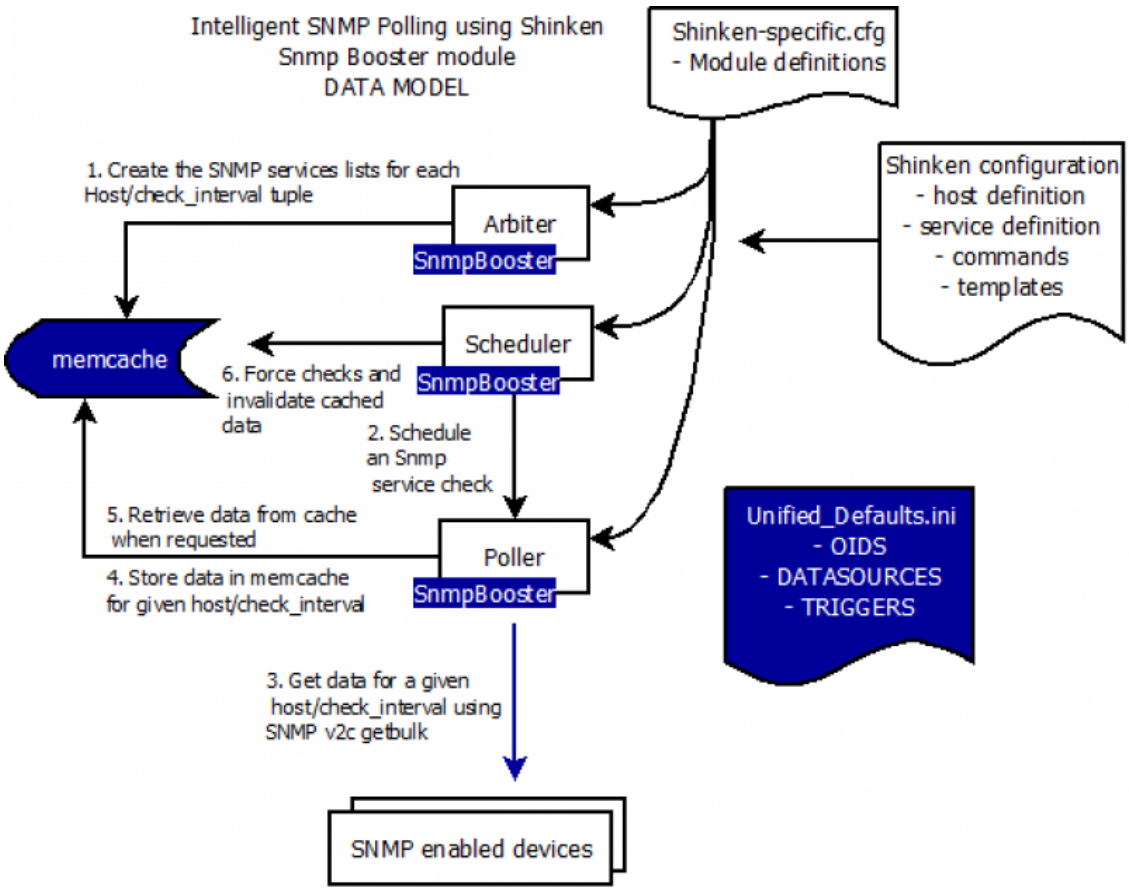

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

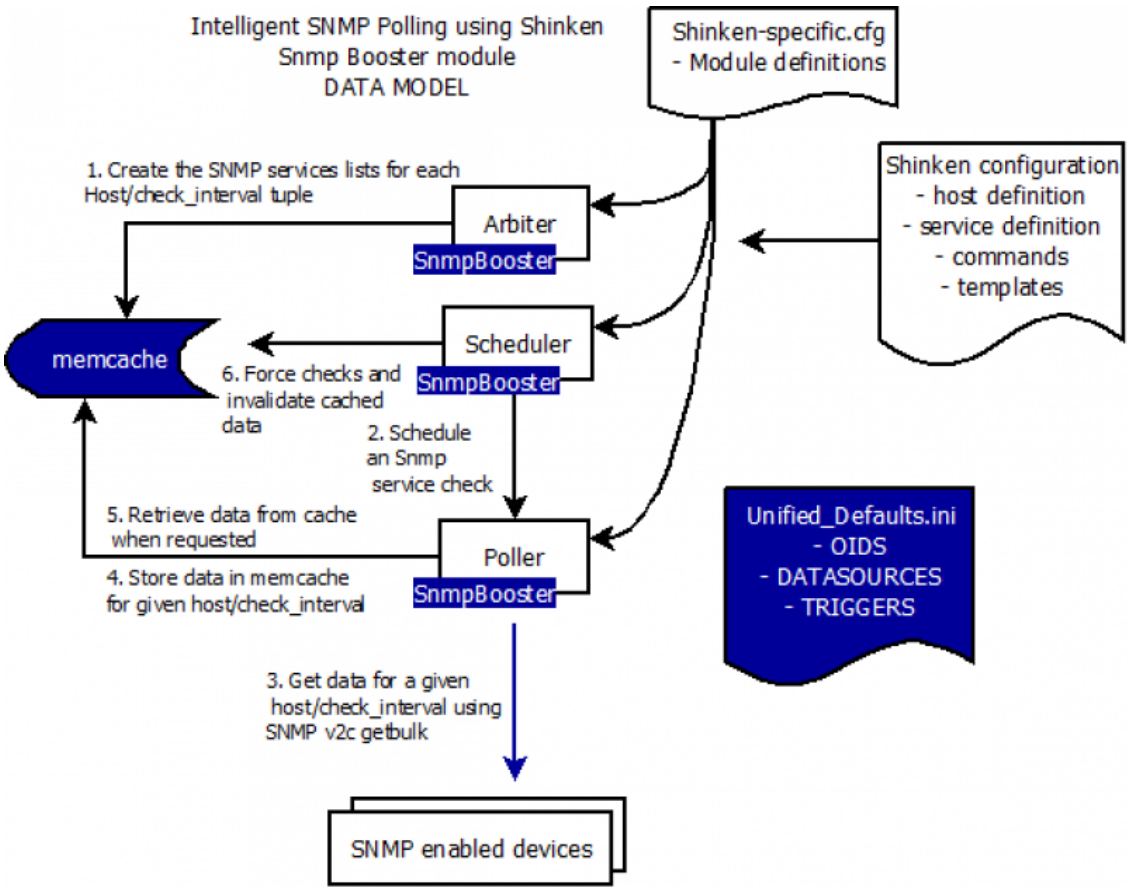

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

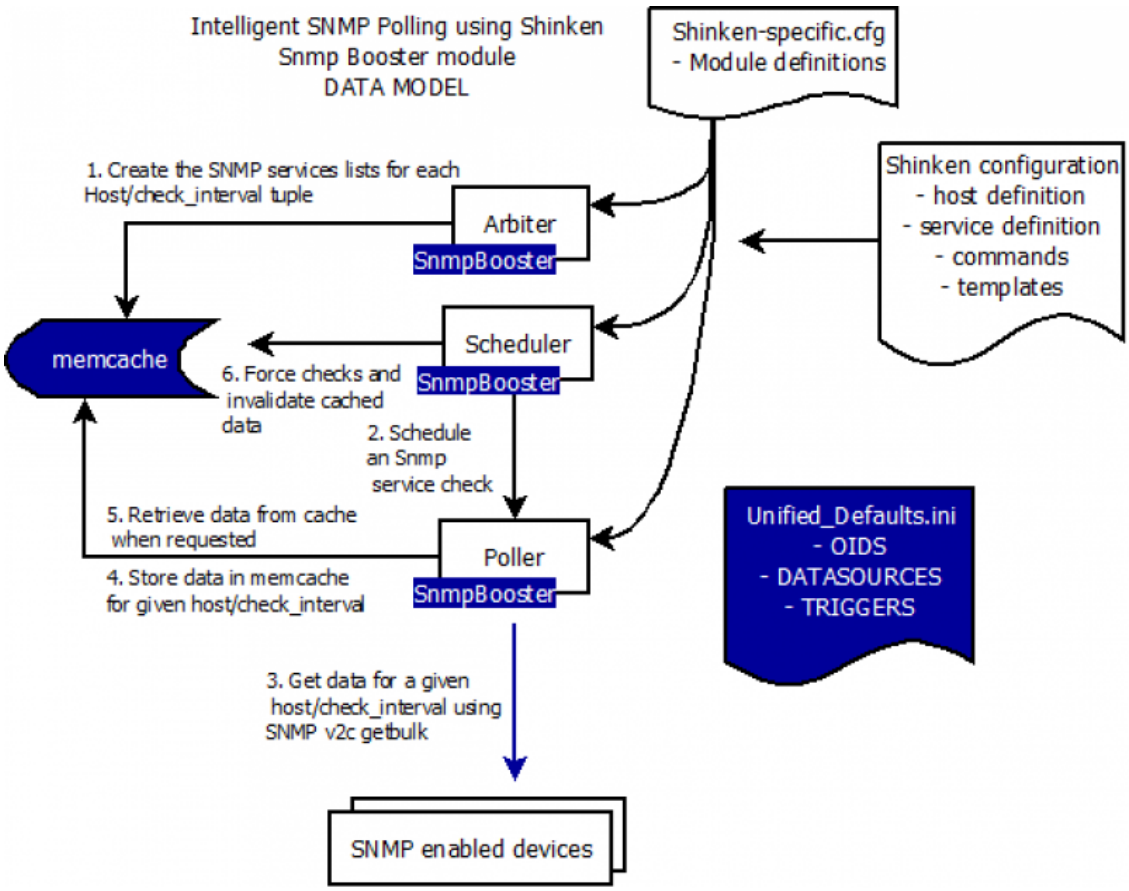

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

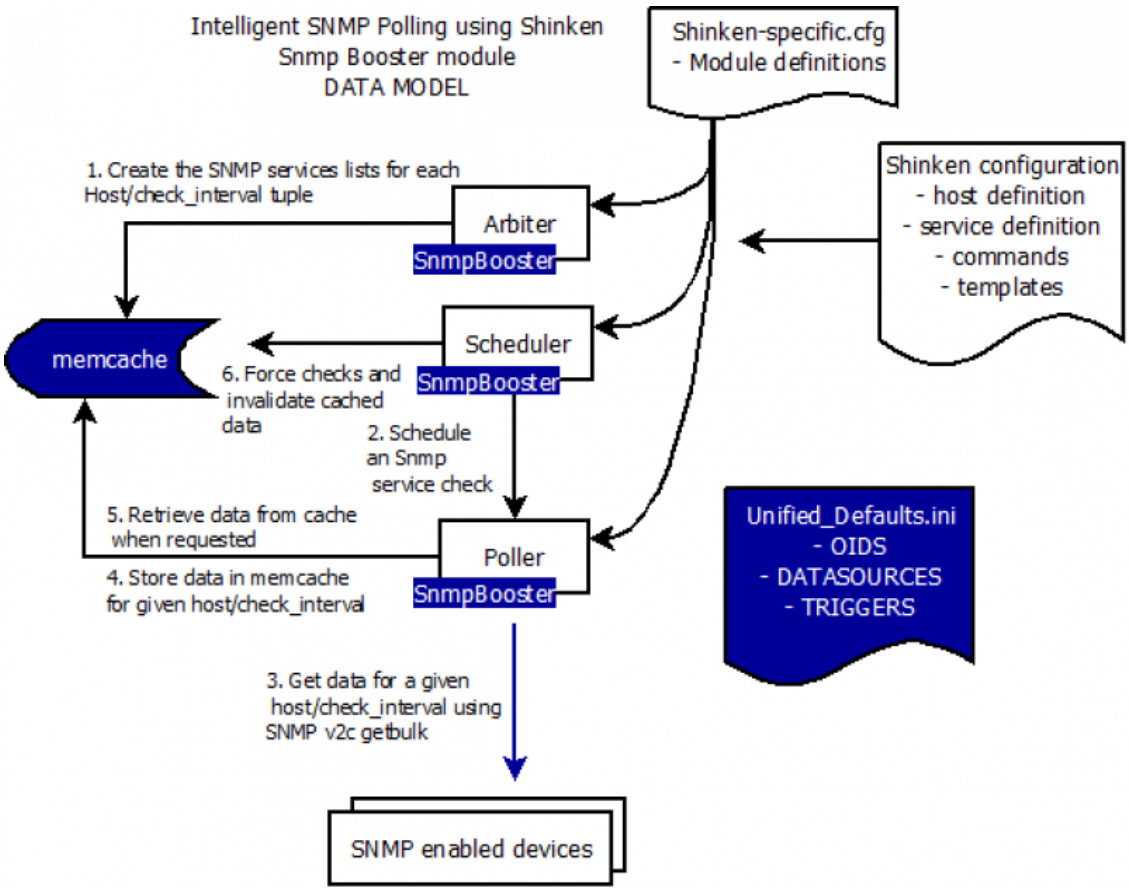

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

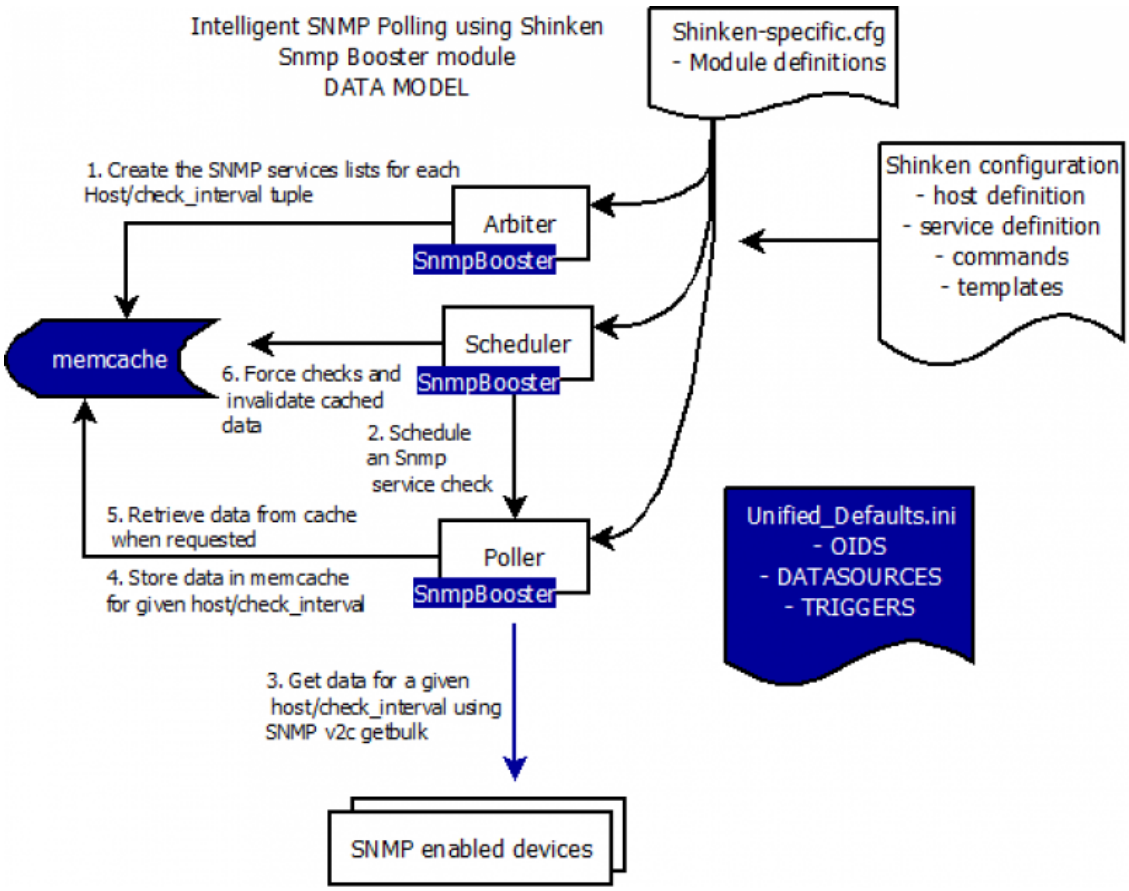

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

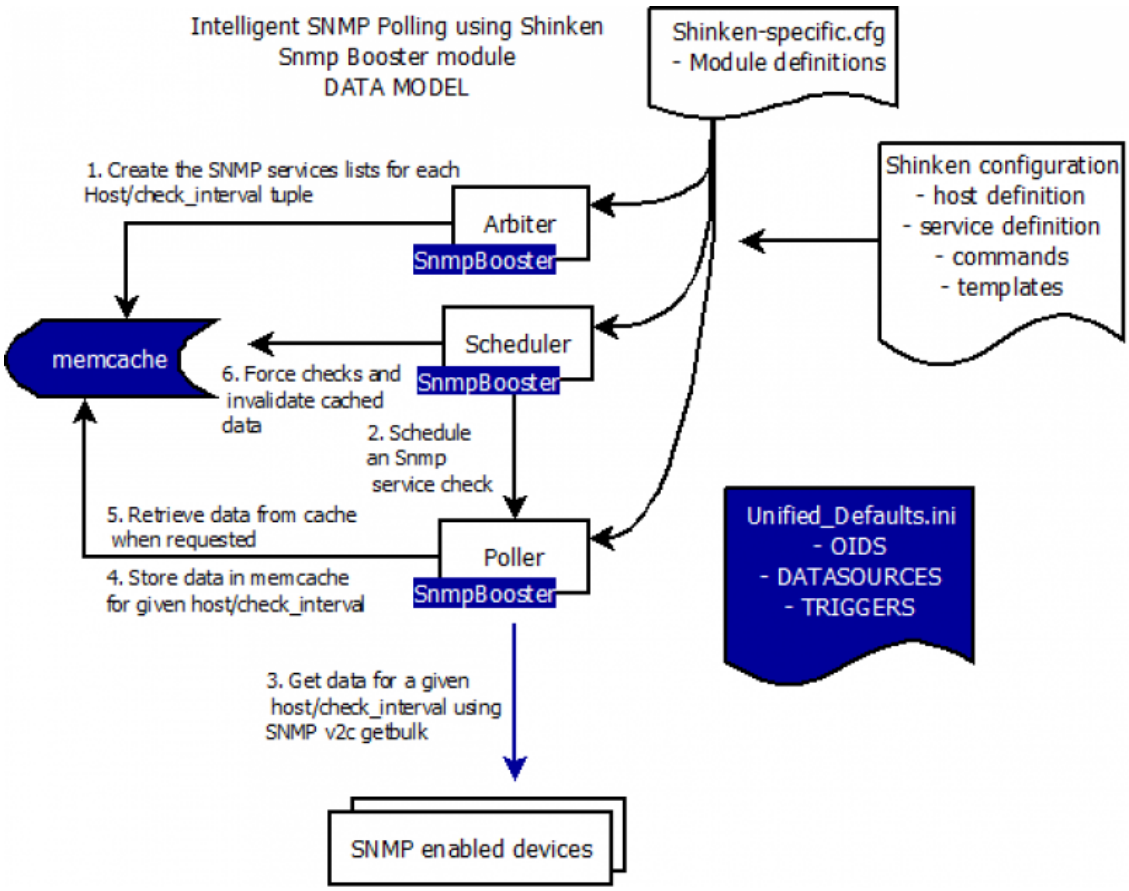

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

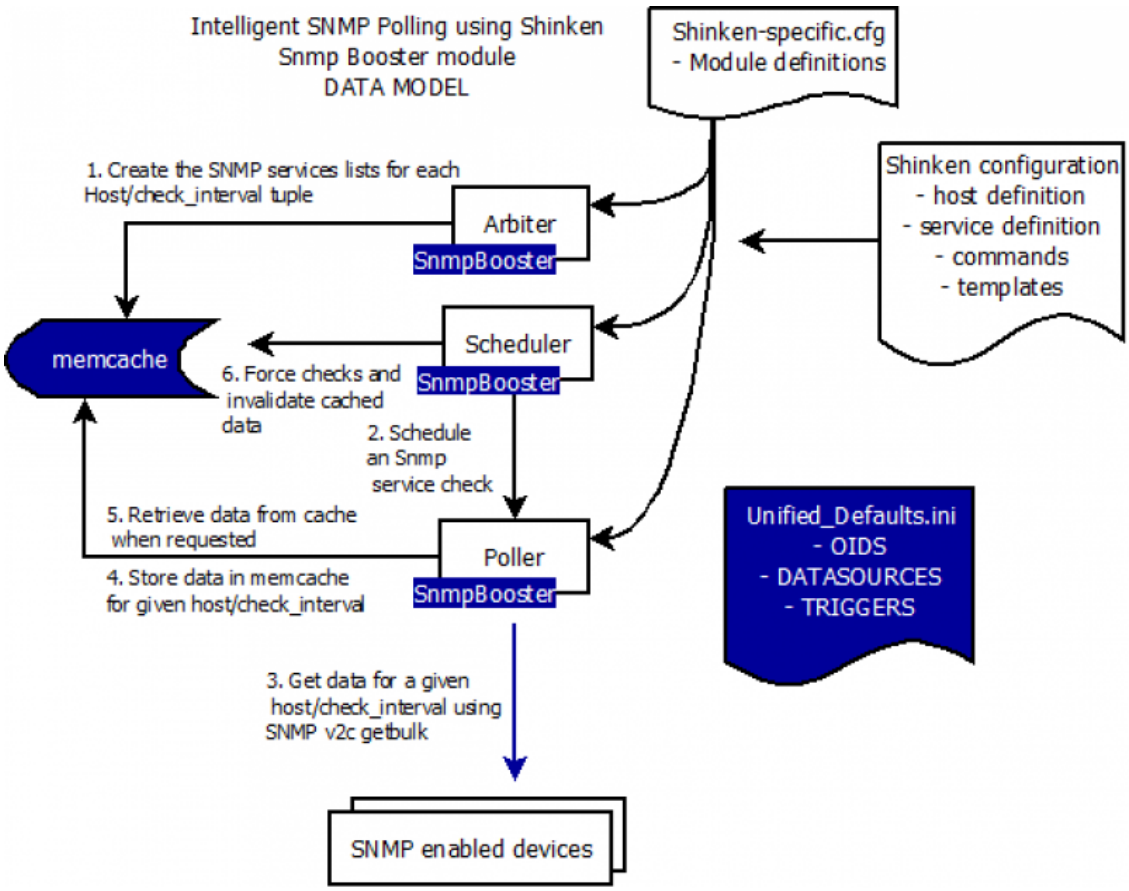

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

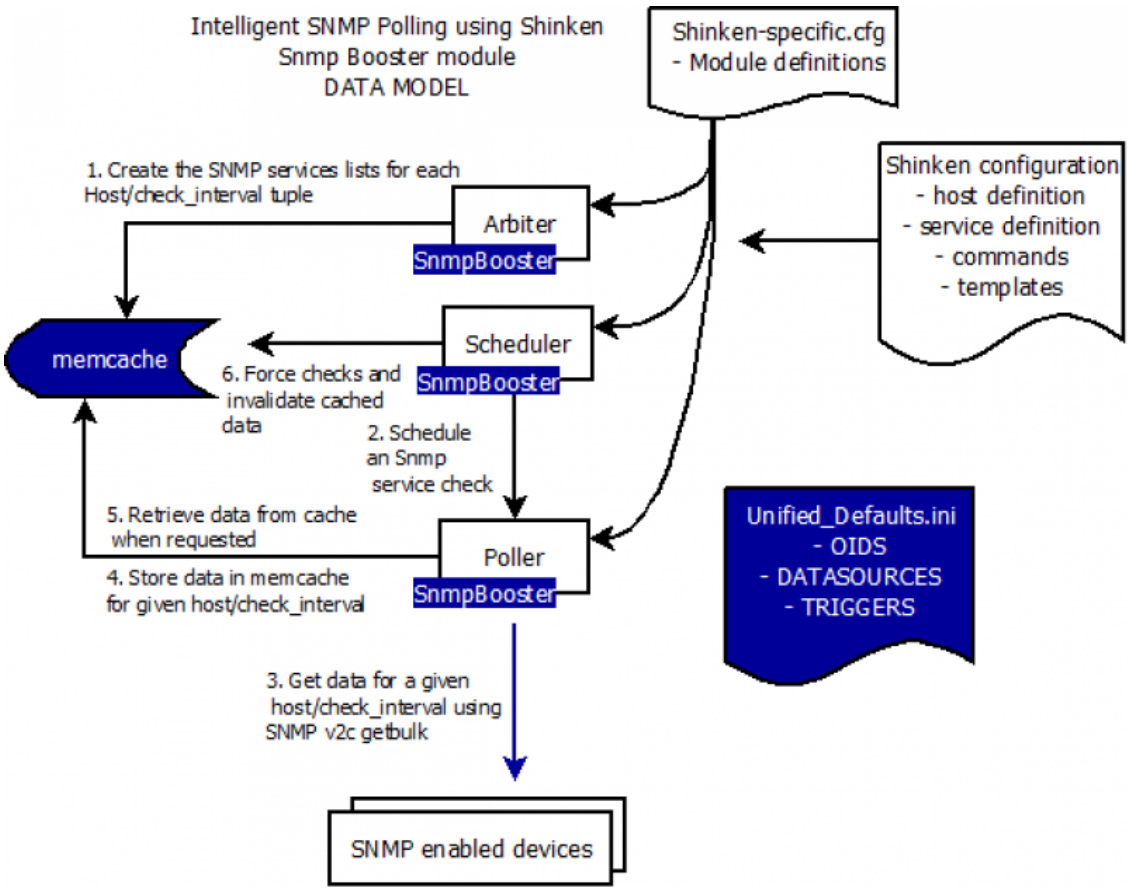

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

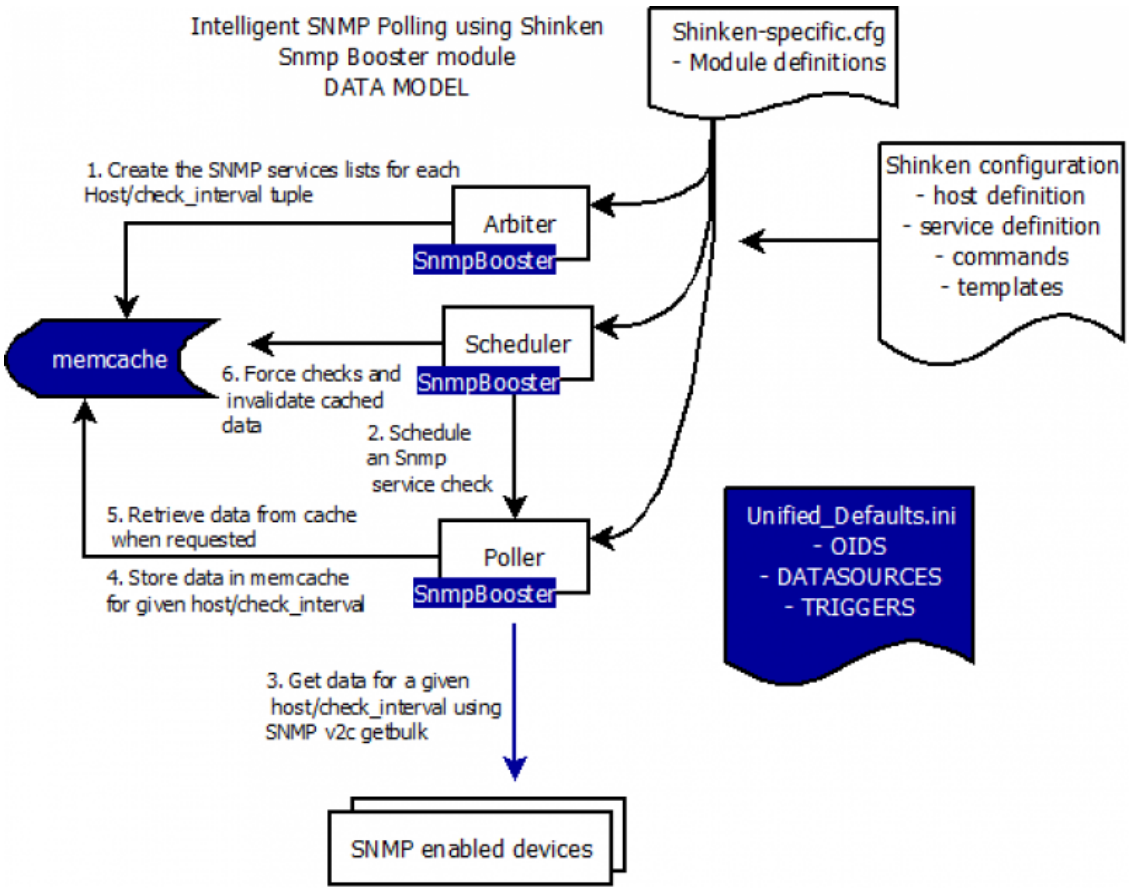

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

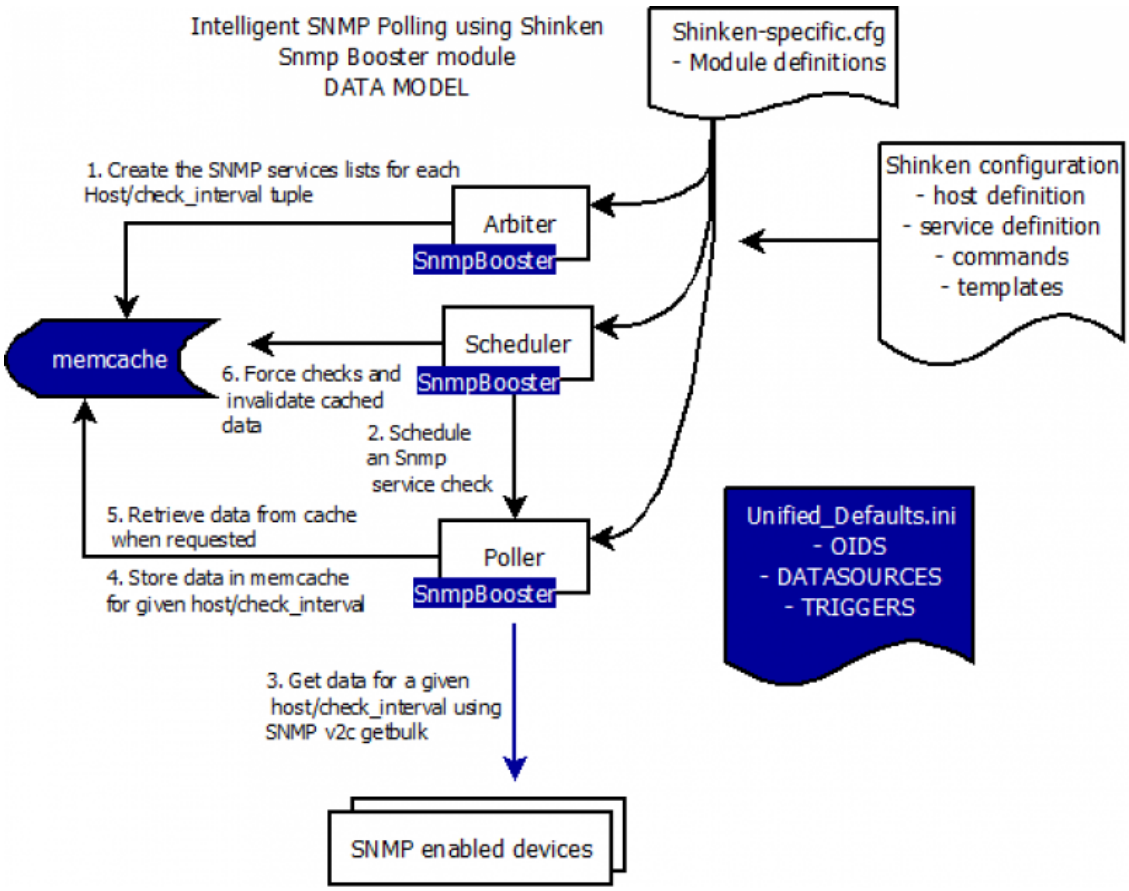

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

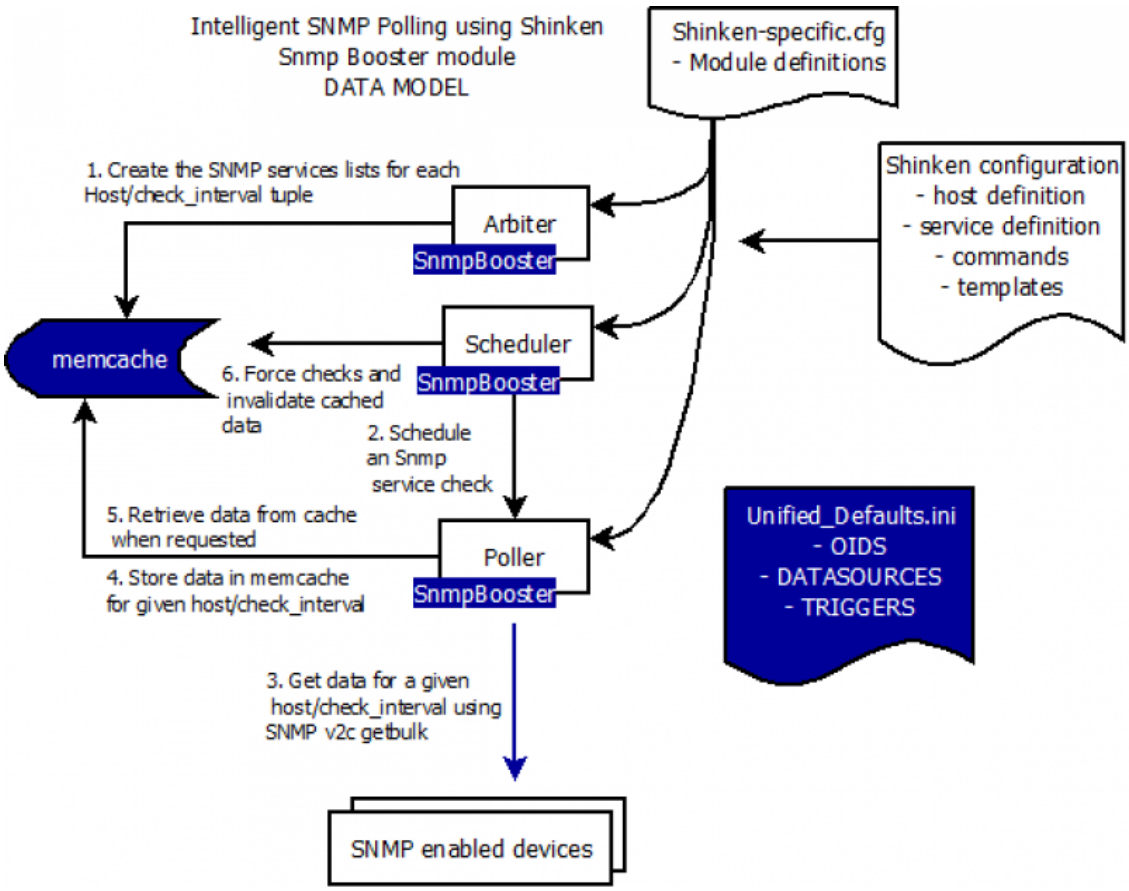

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

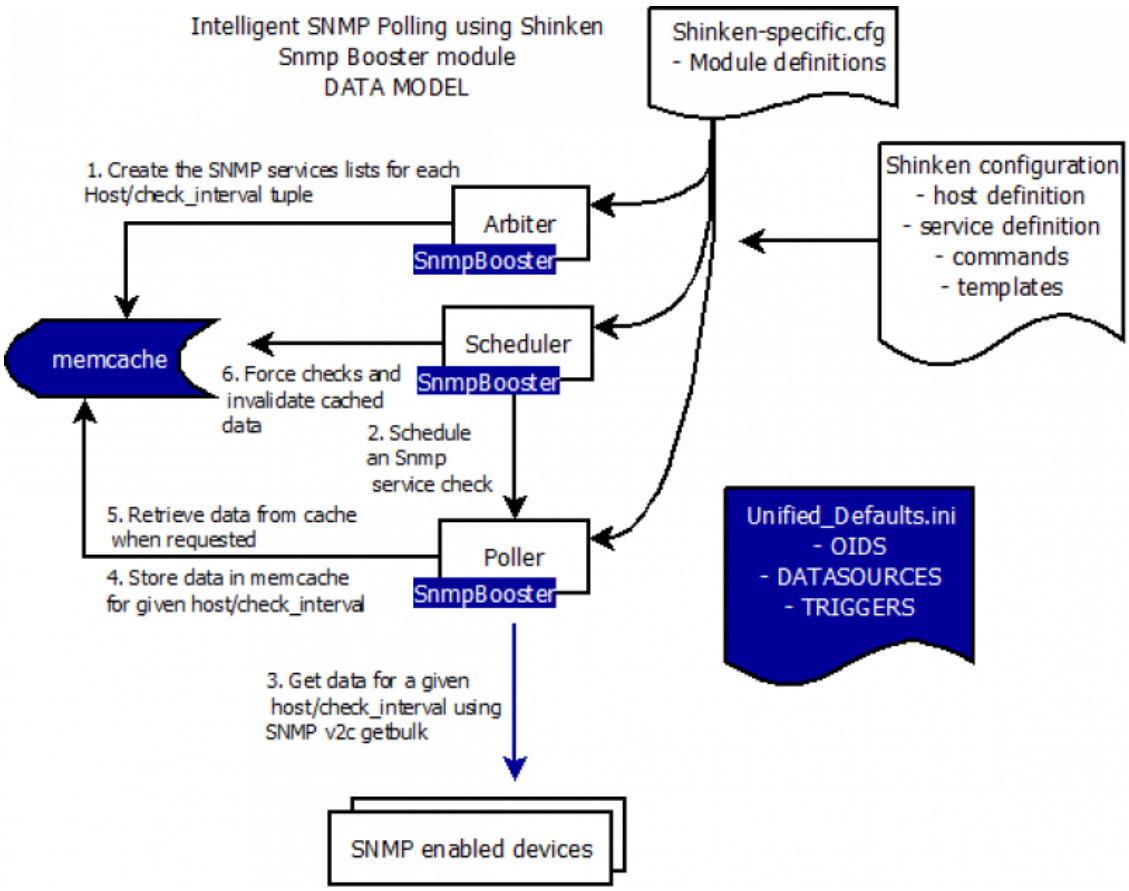

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

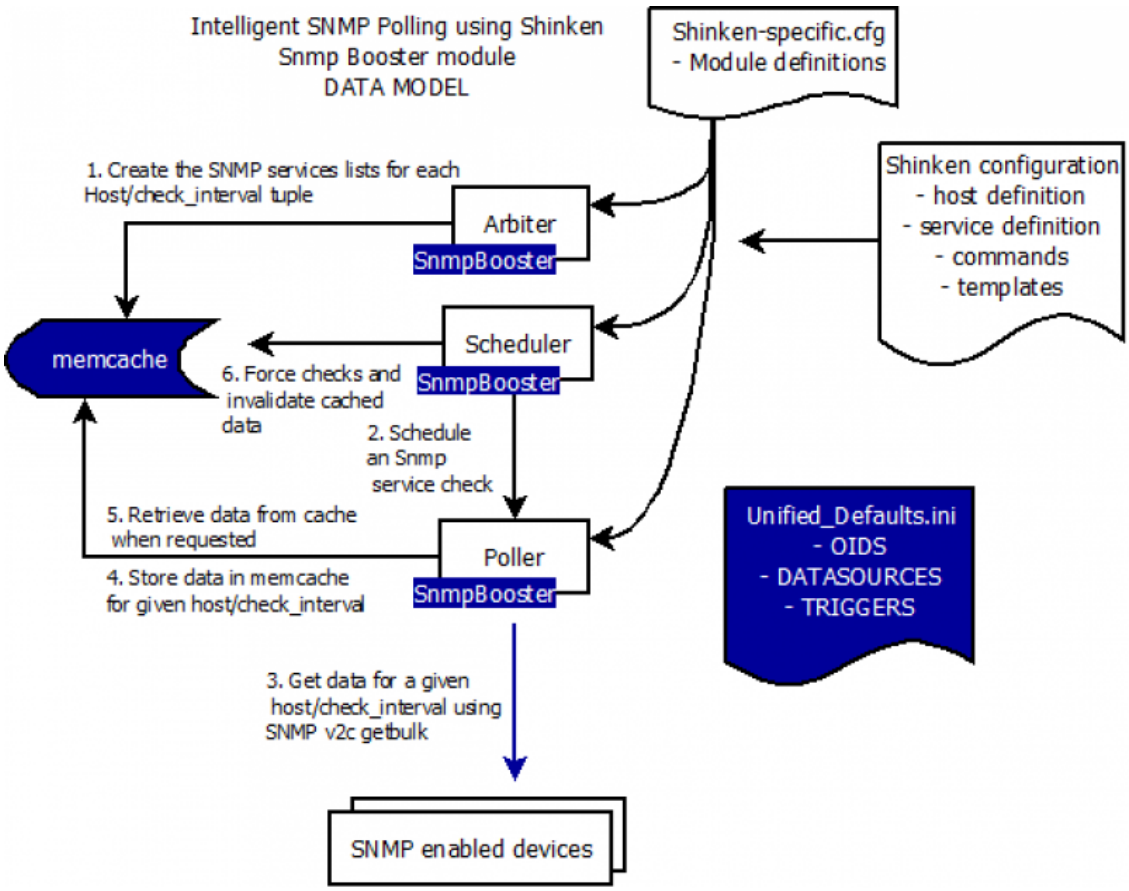

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

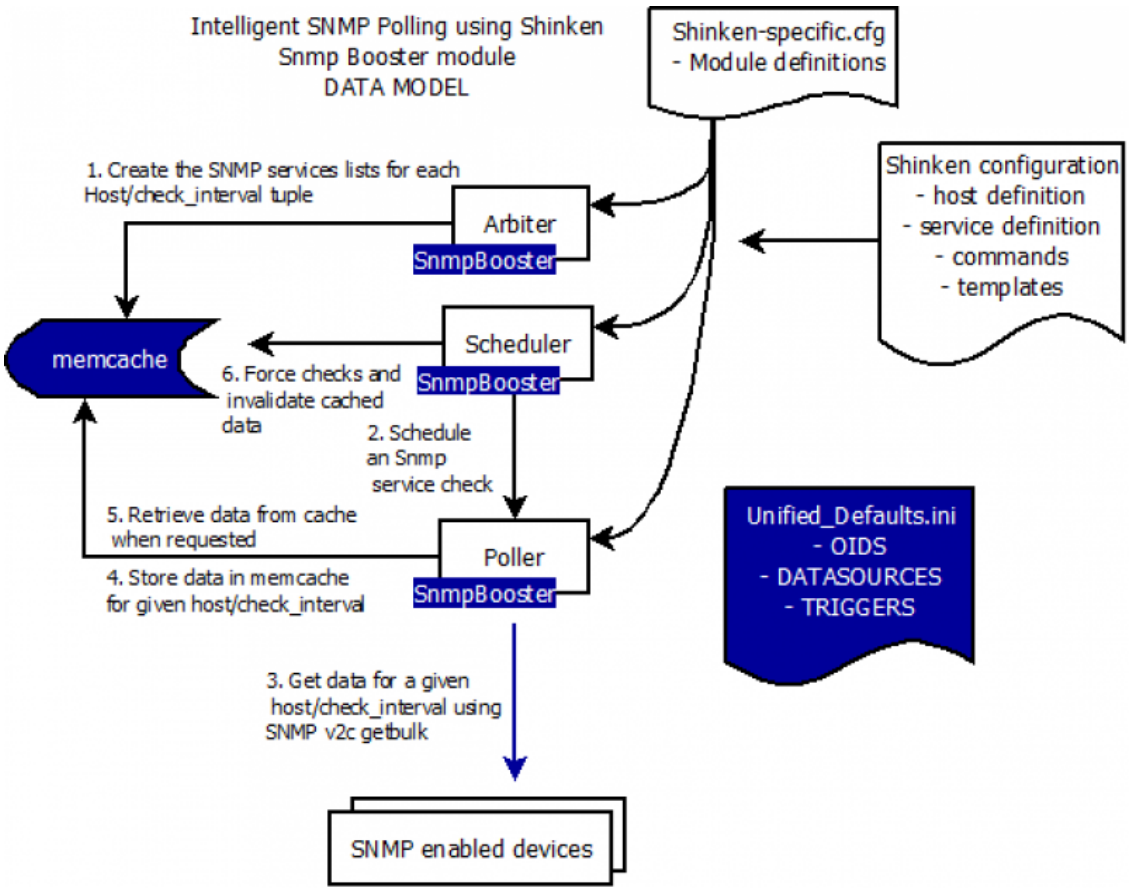

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

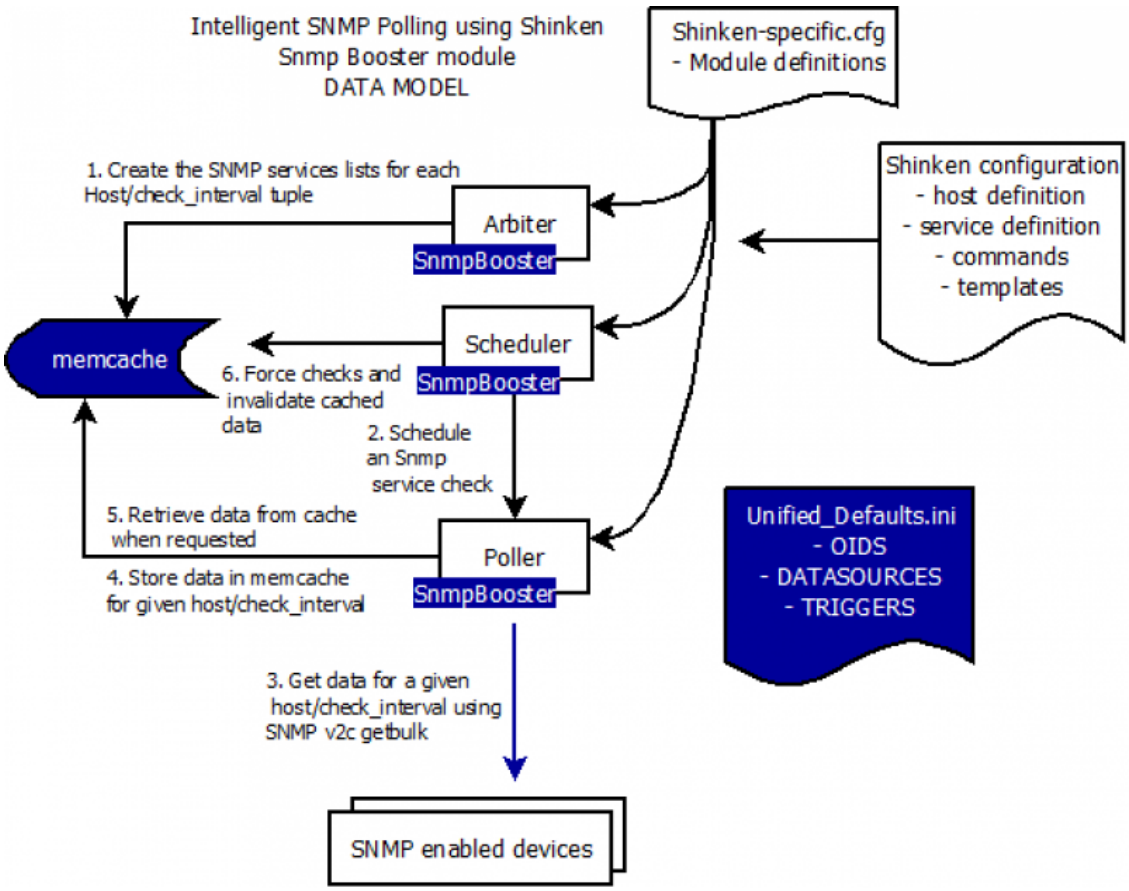

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

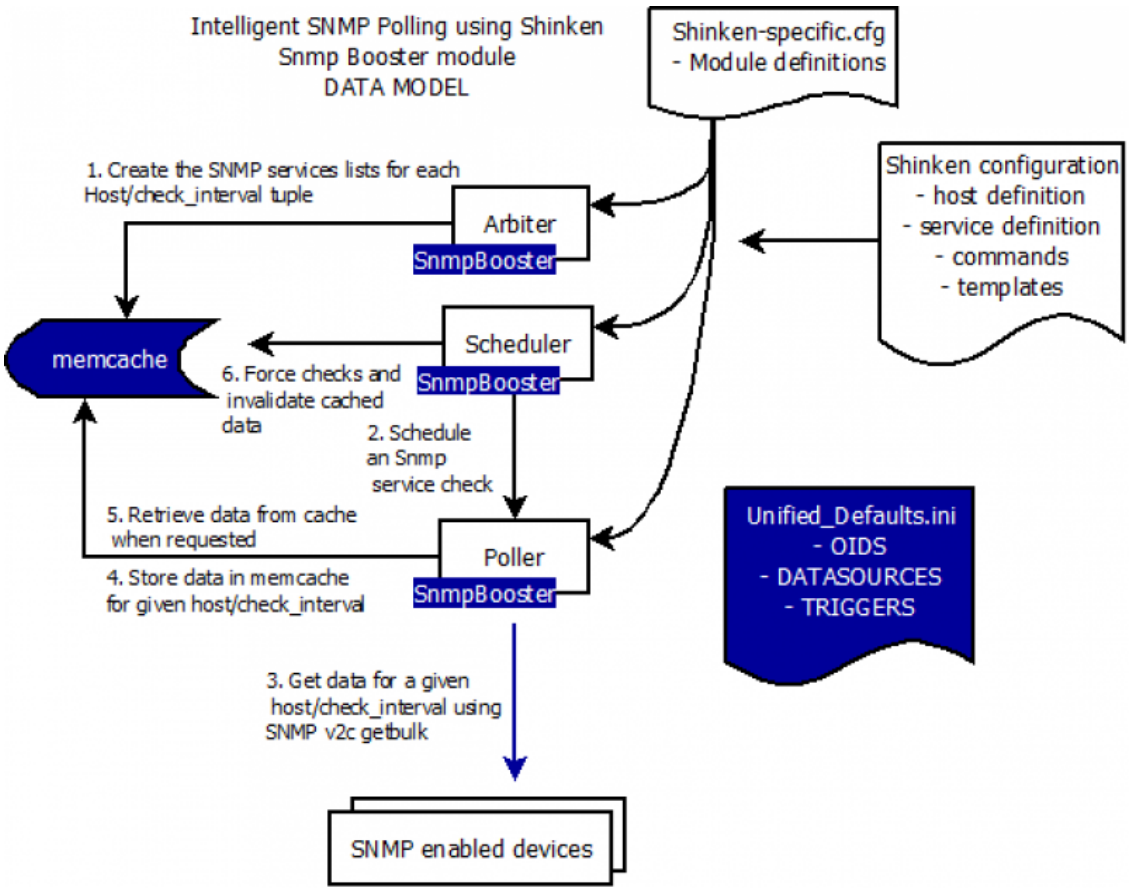

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

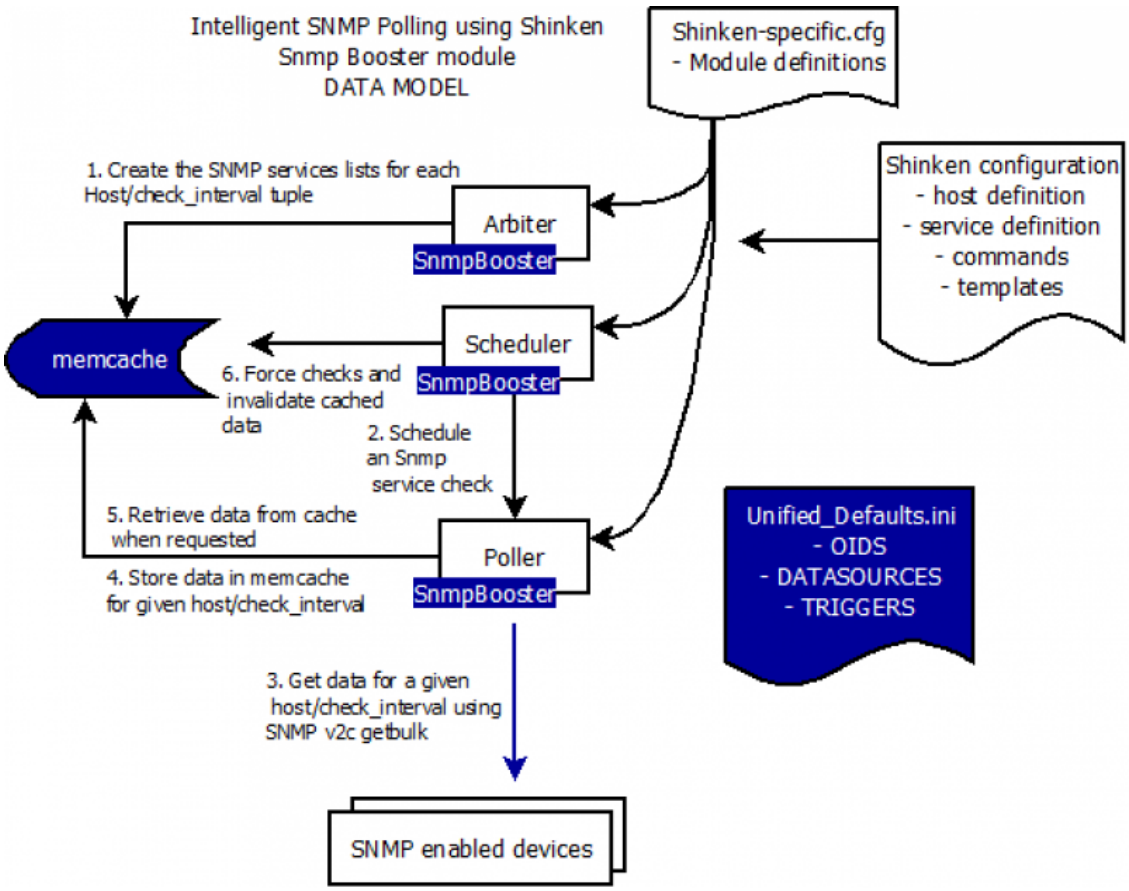

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

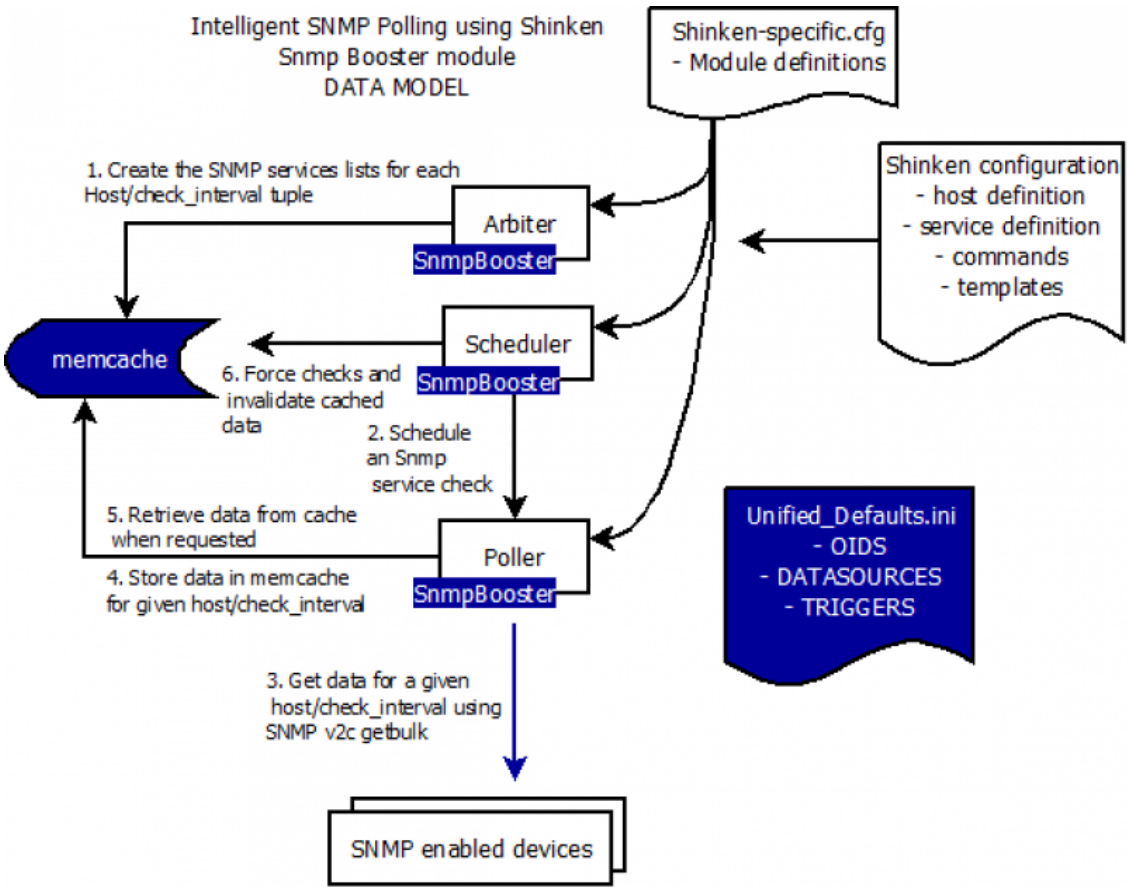

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

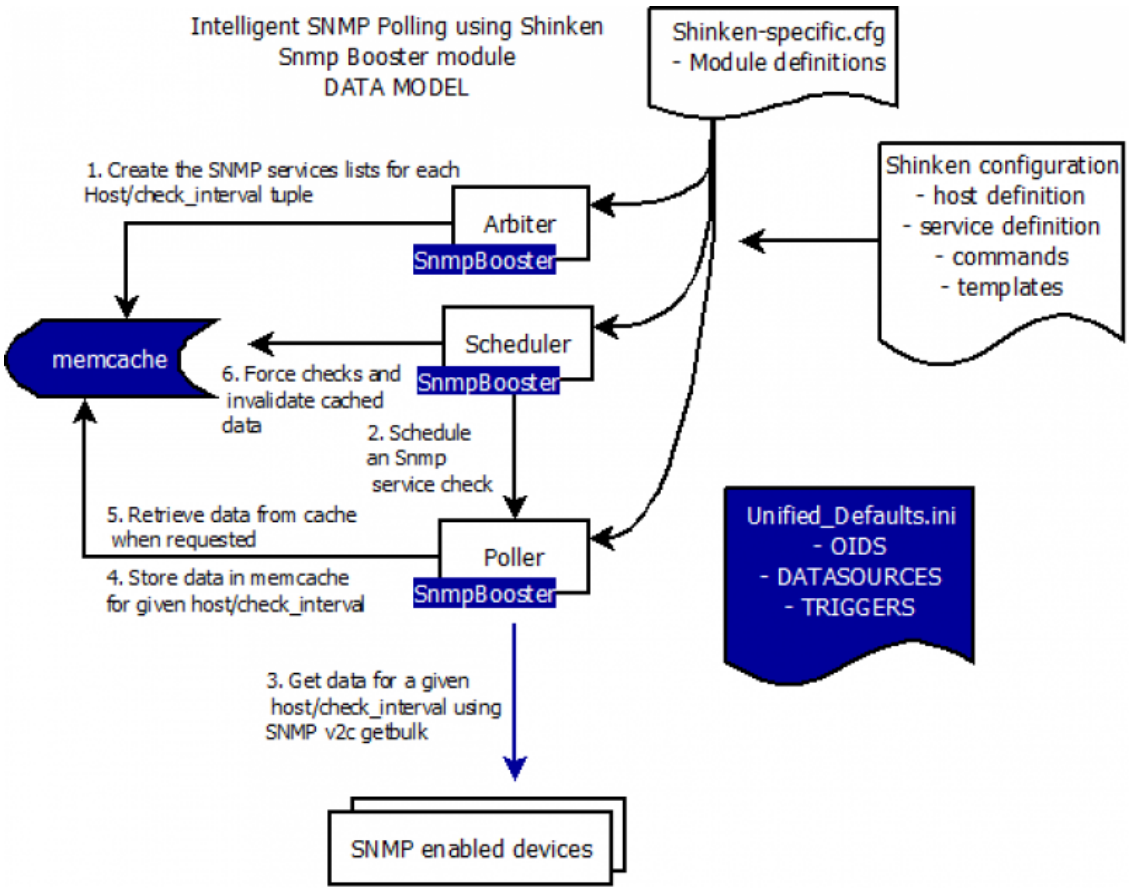

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

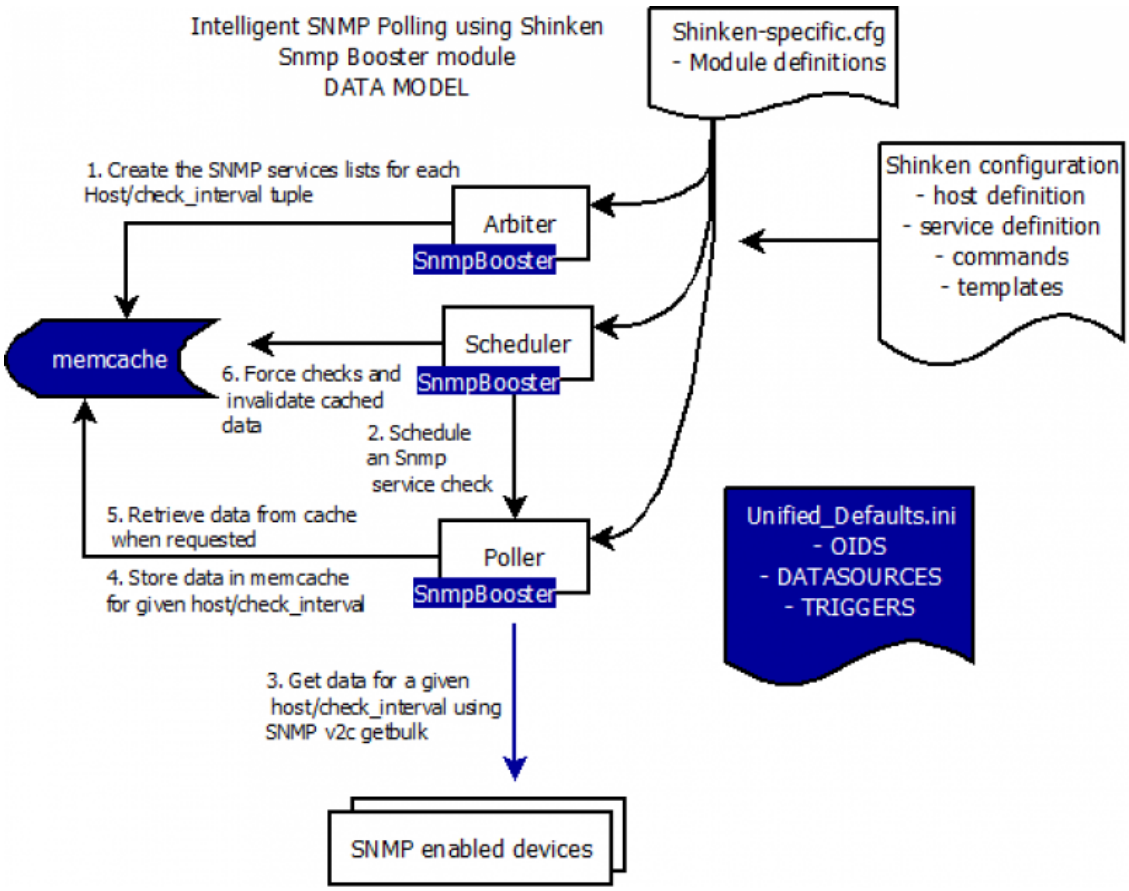

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

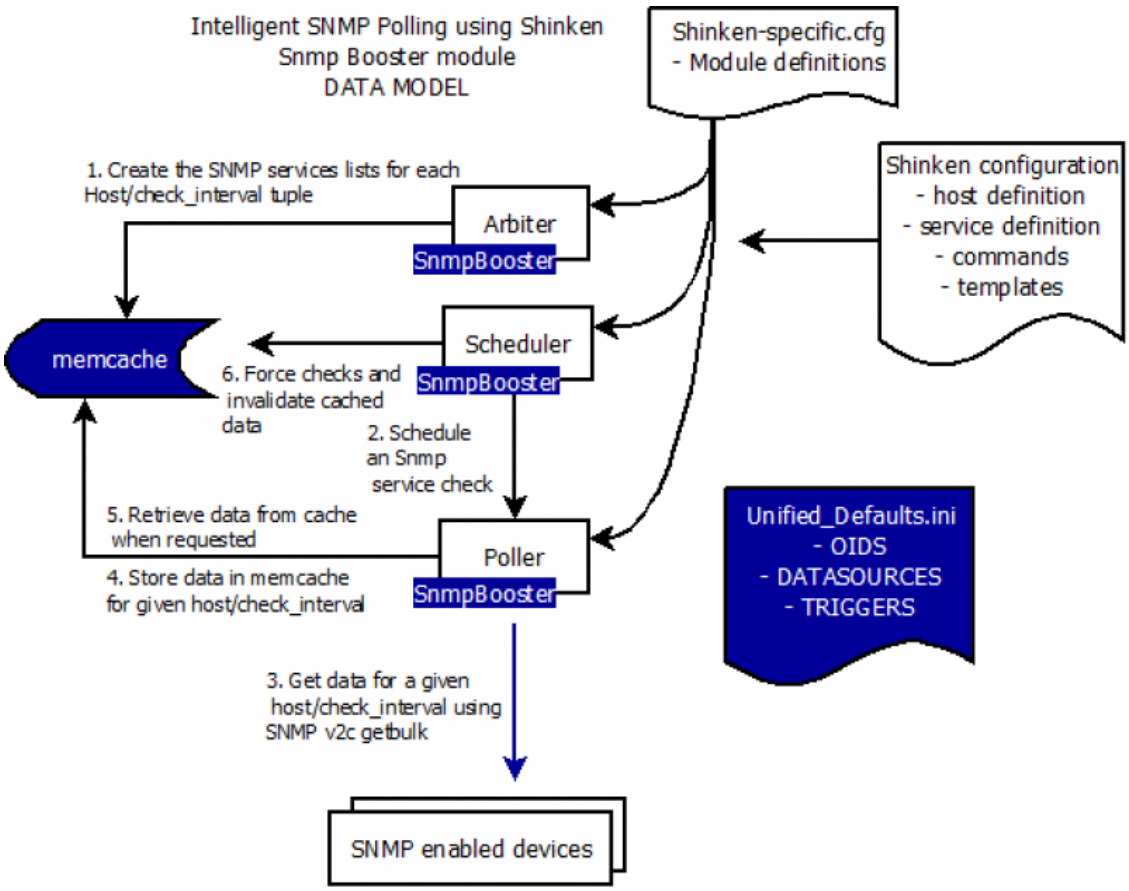

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'**_):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

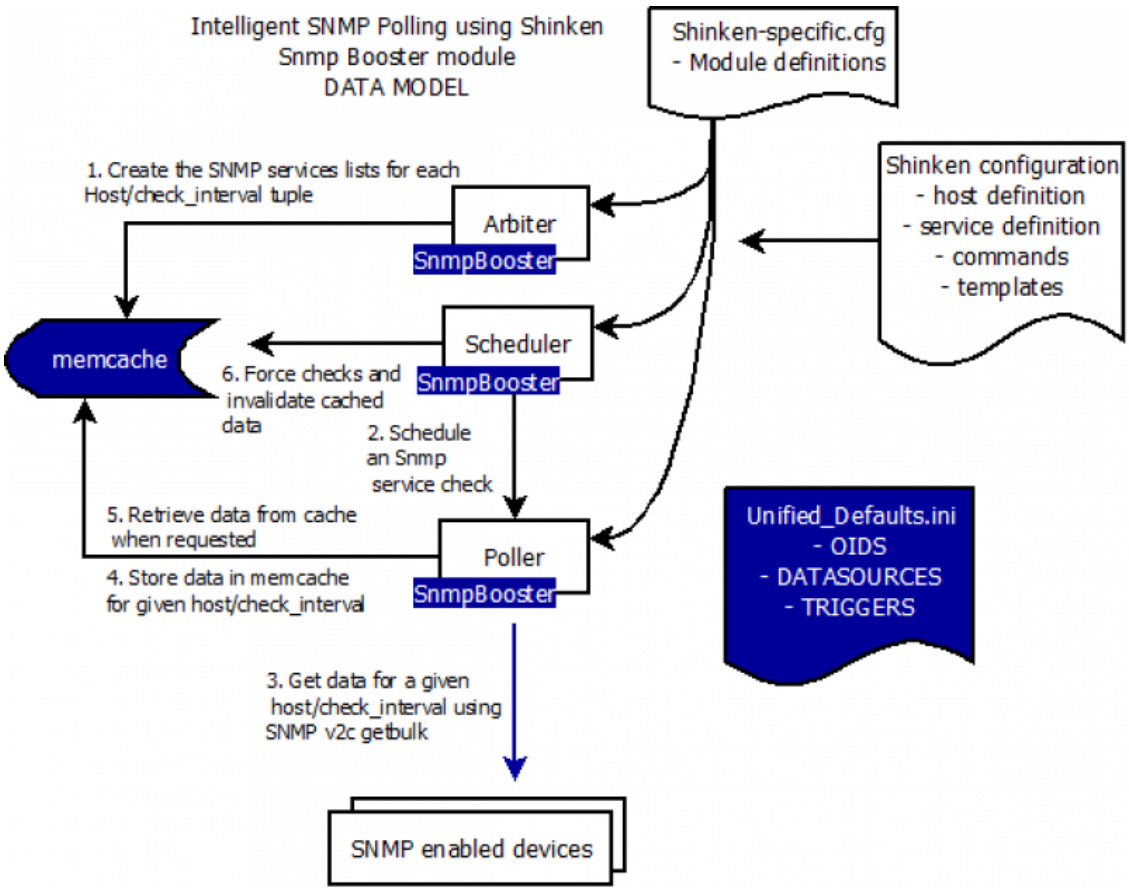

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

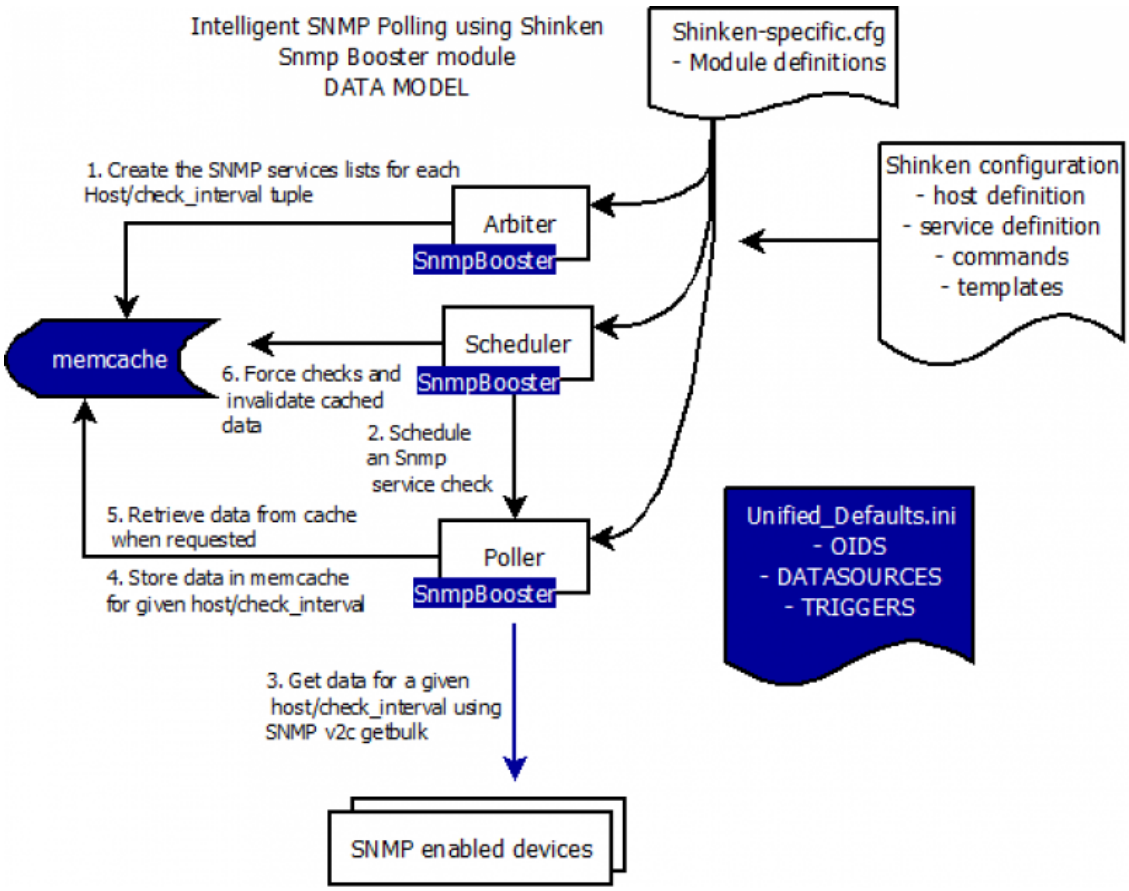

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

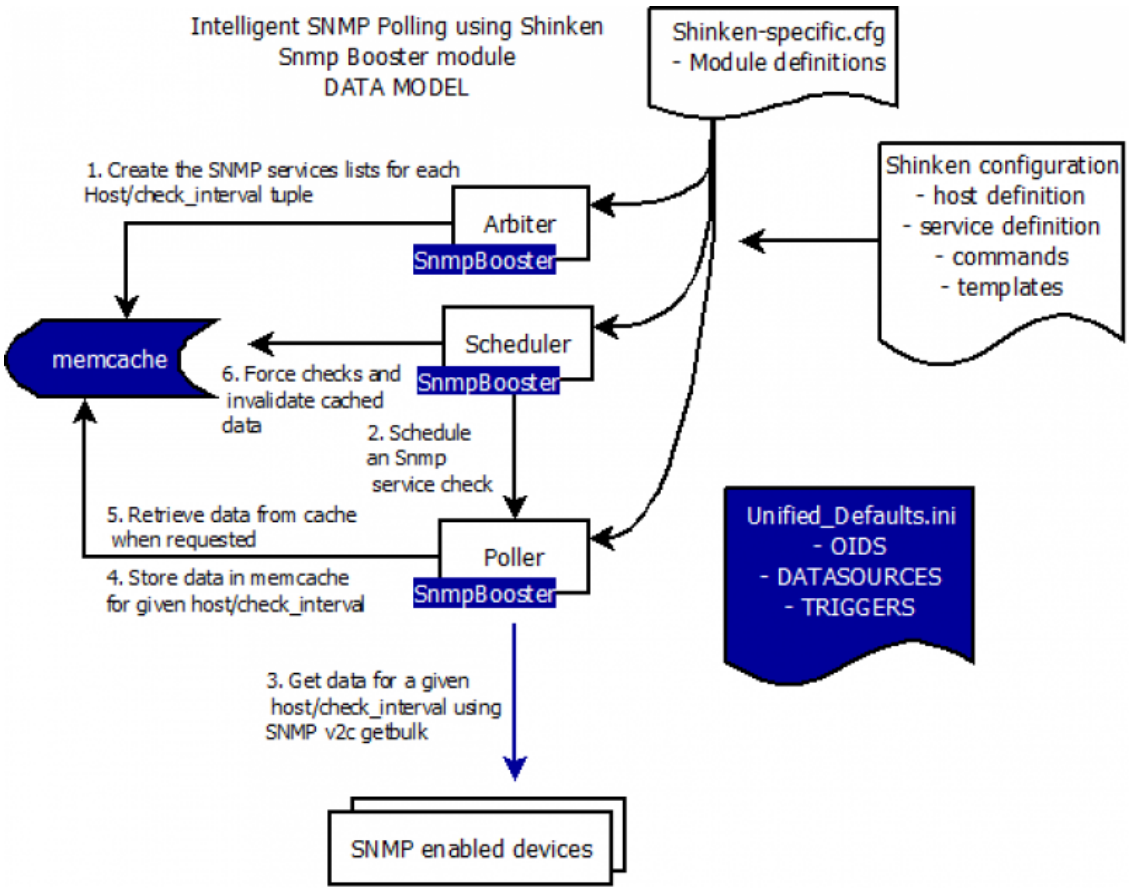

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

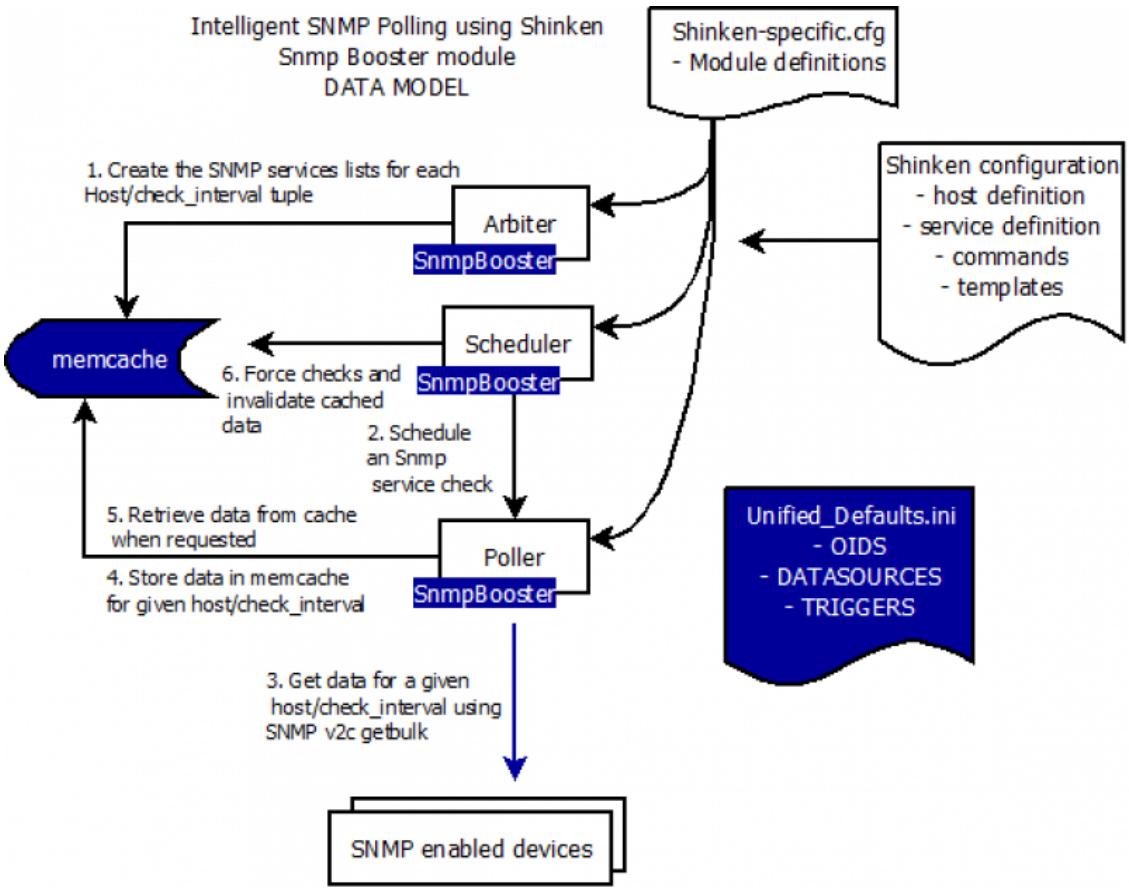

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

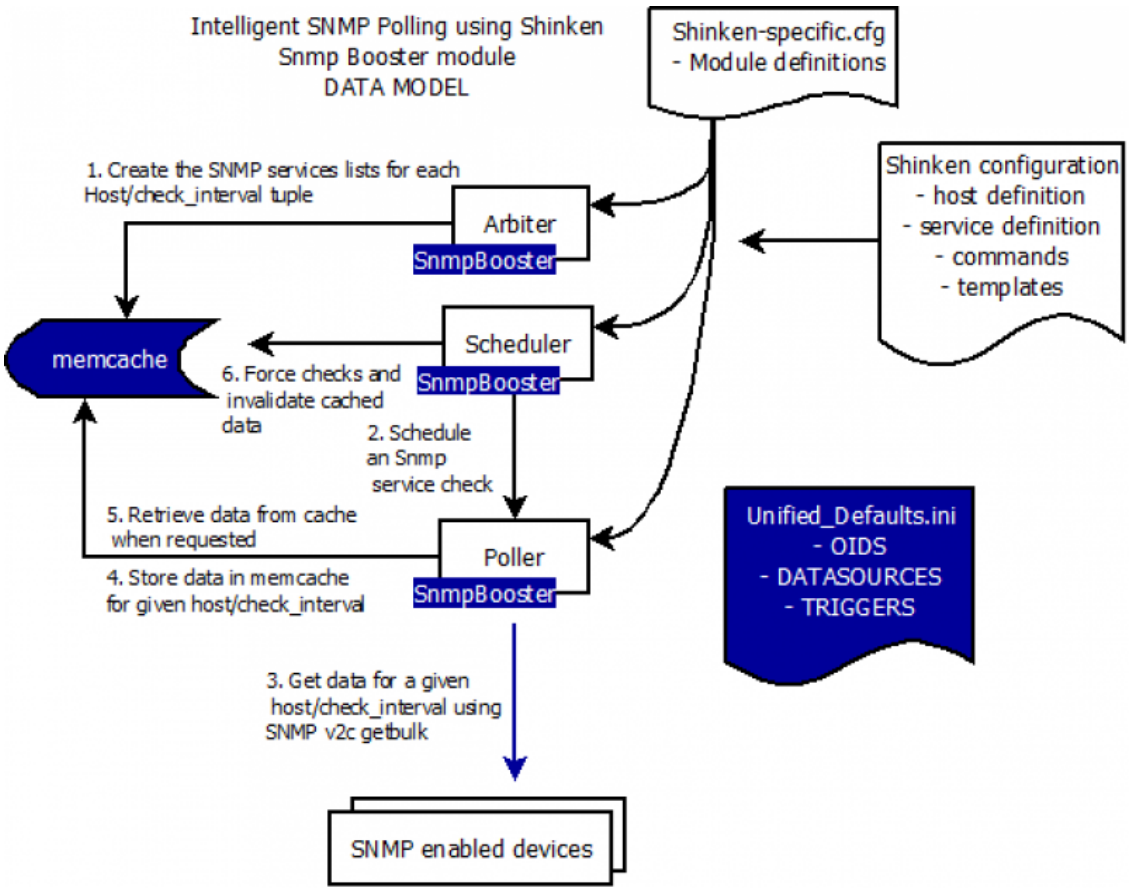

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

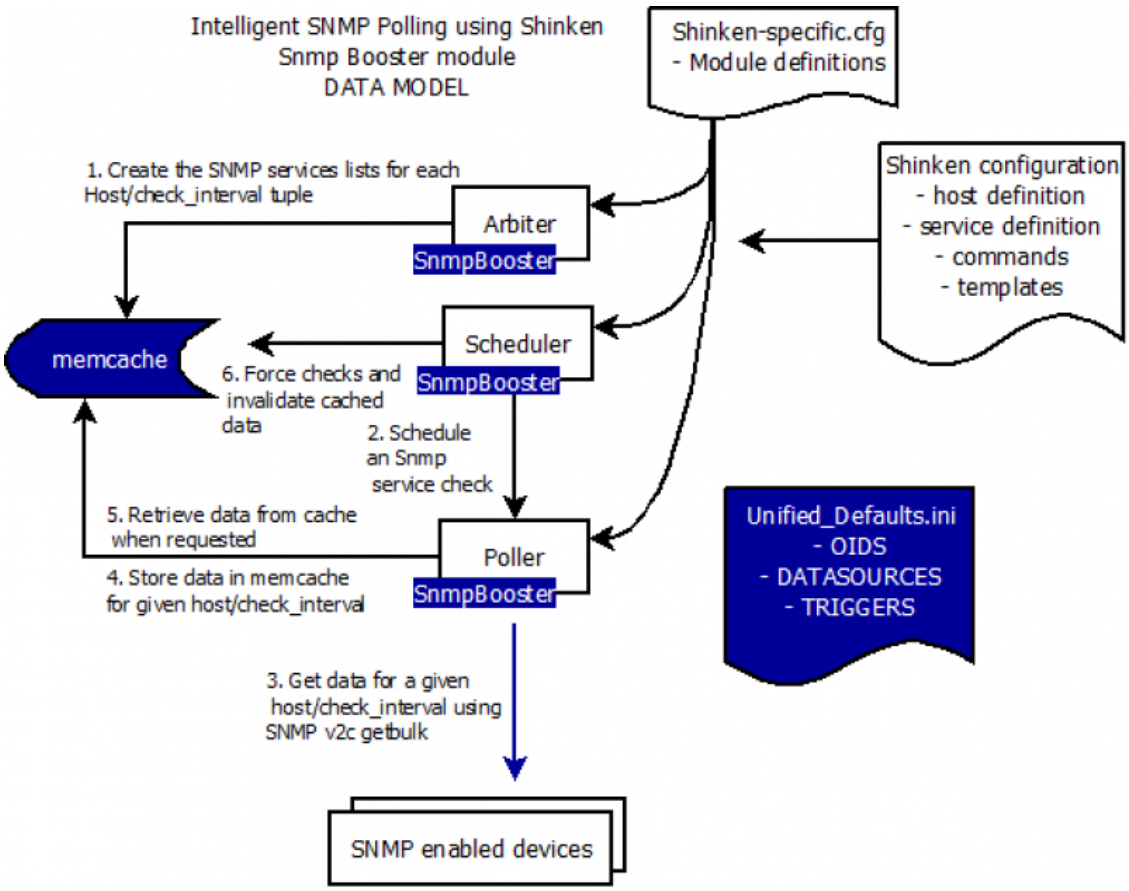

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

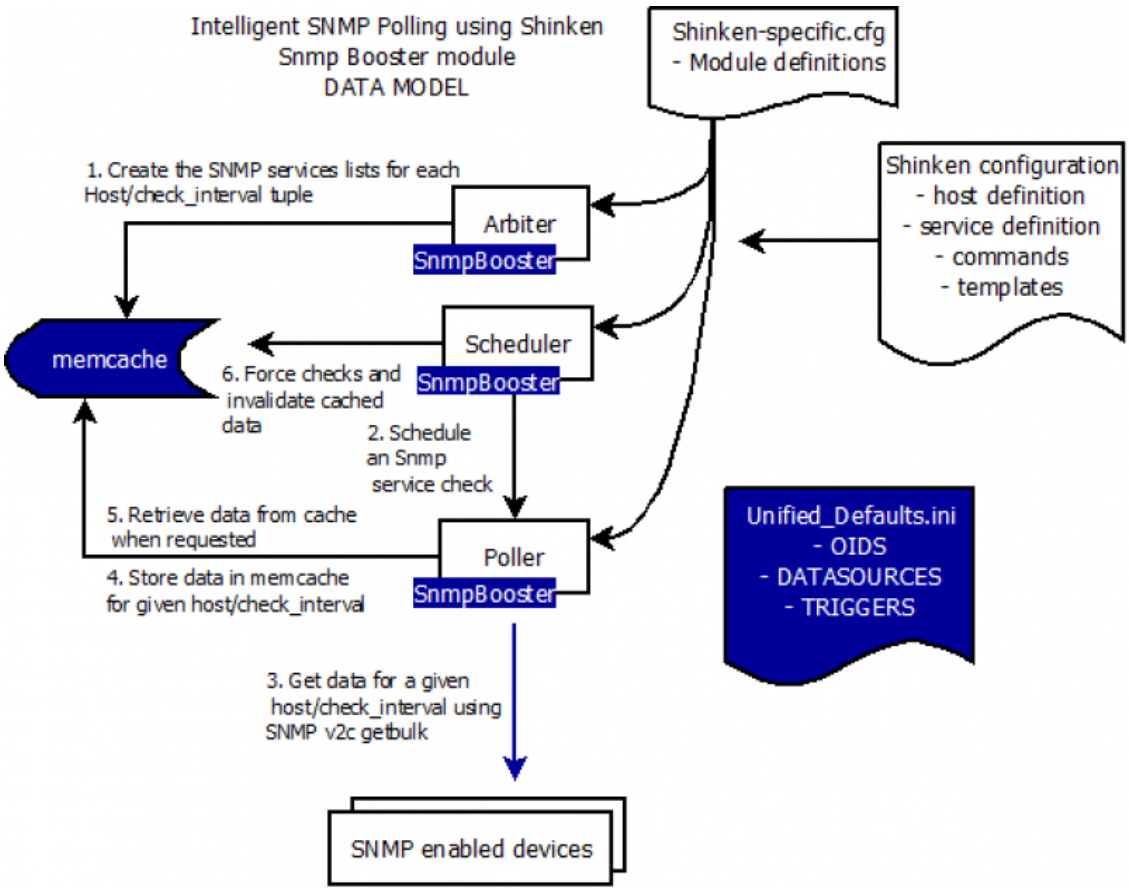

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

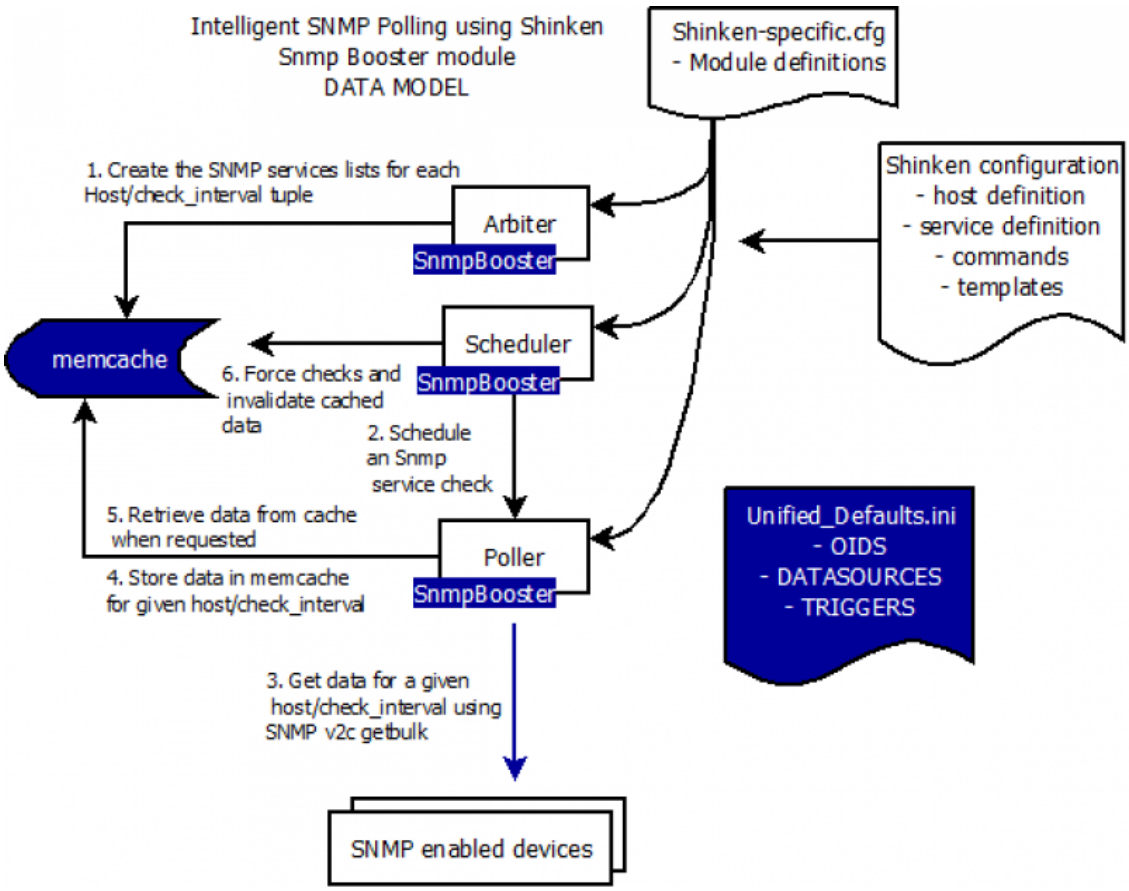

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

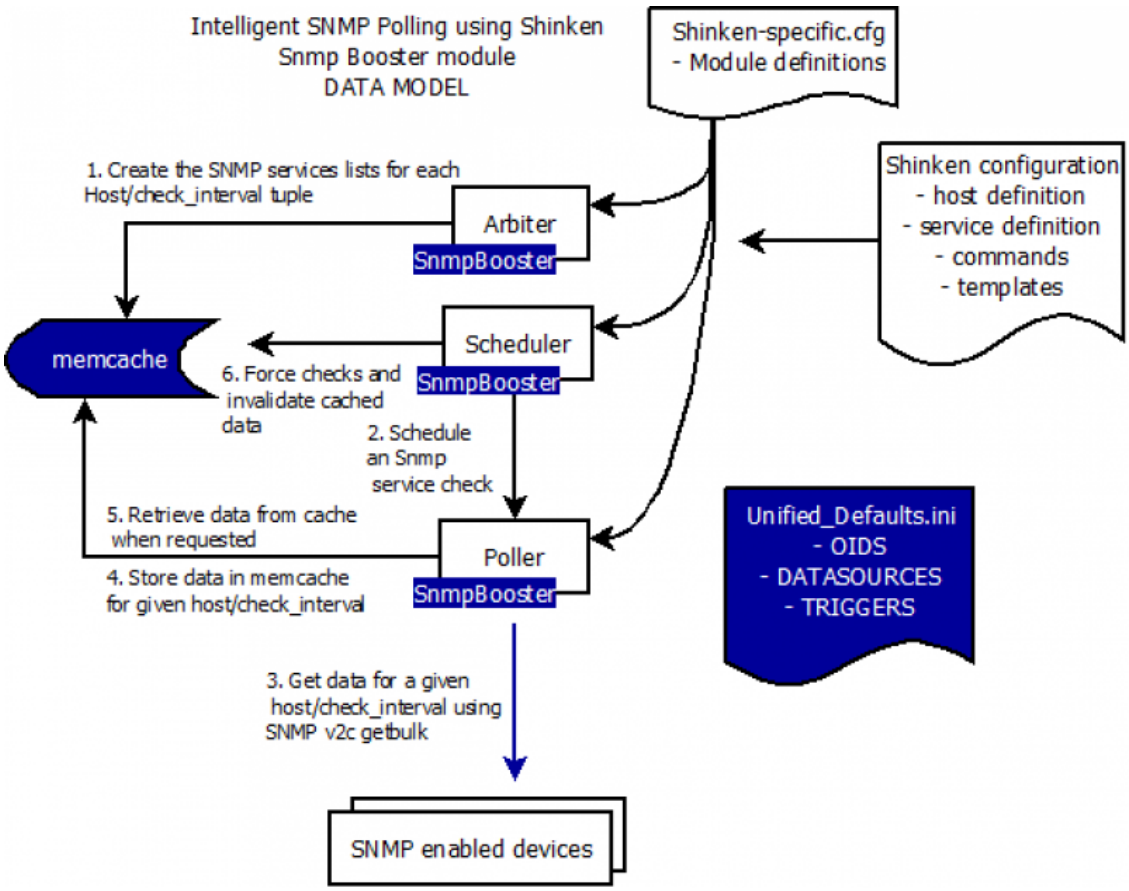

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

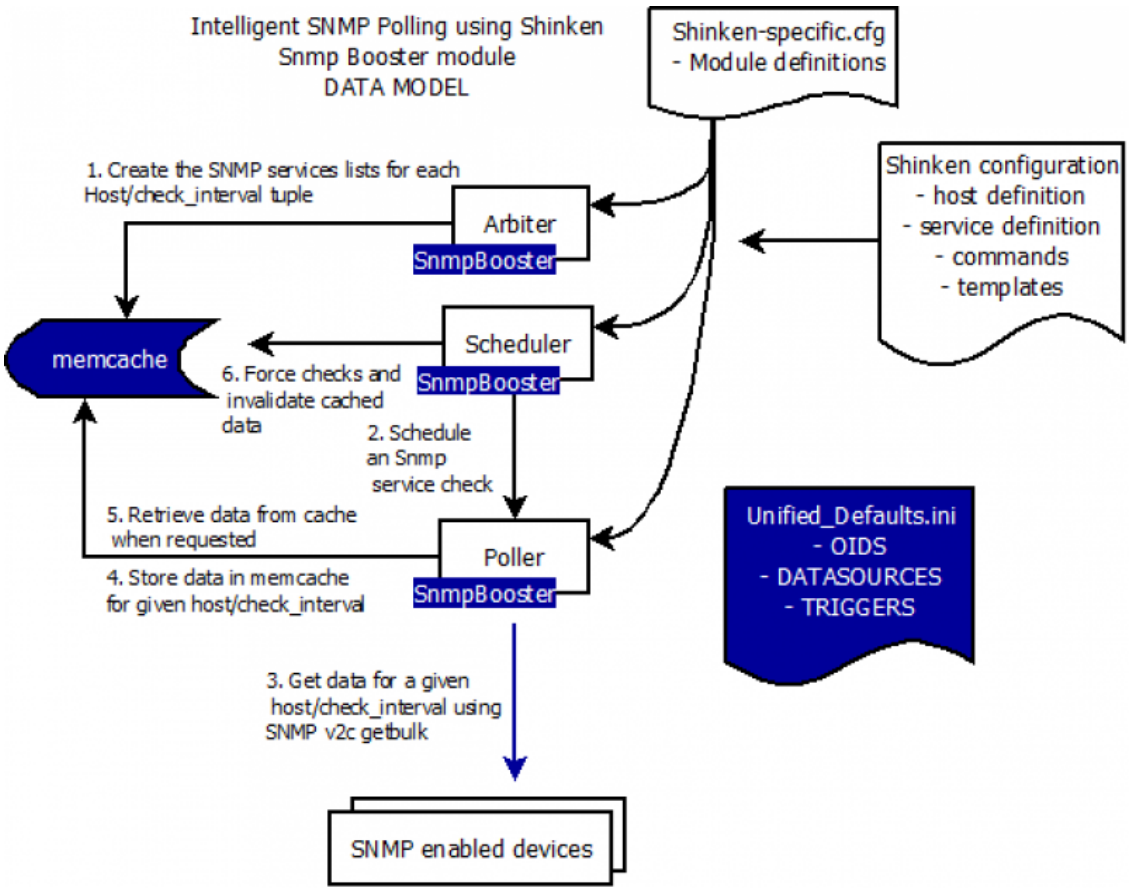

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

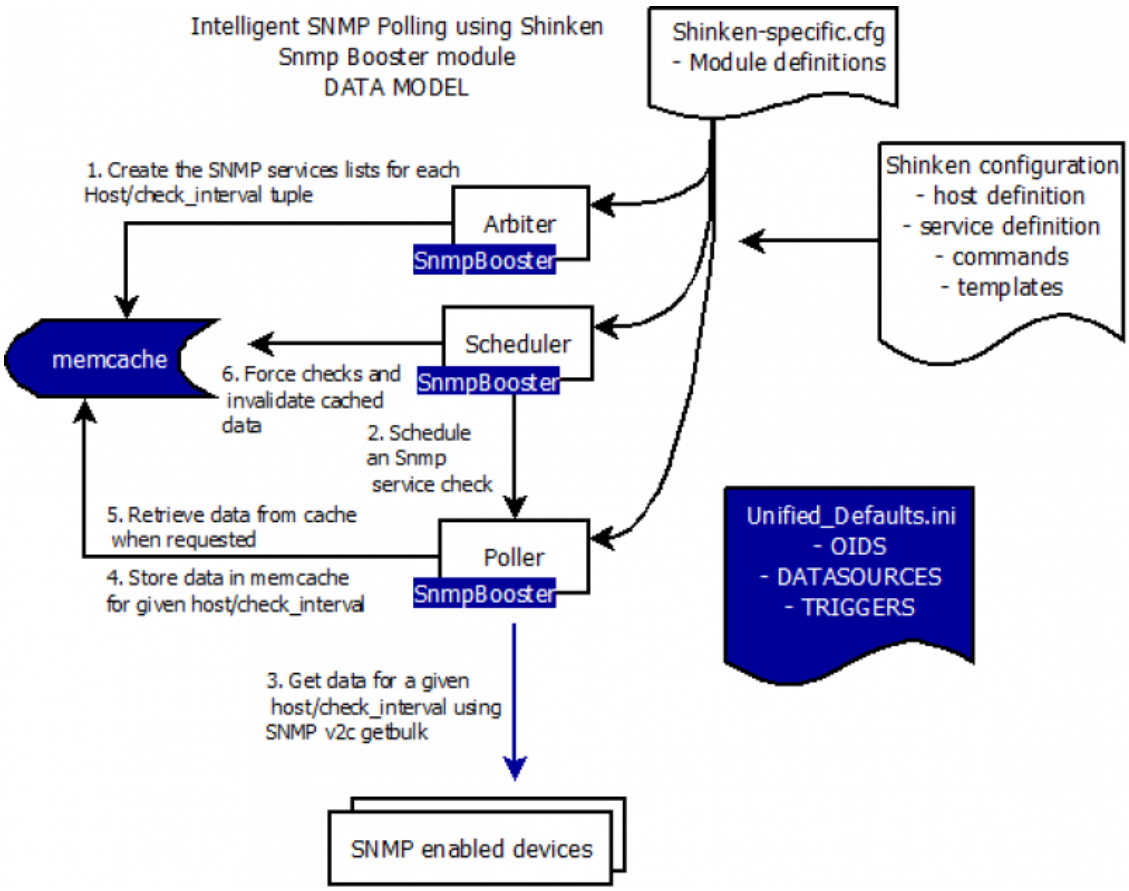

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

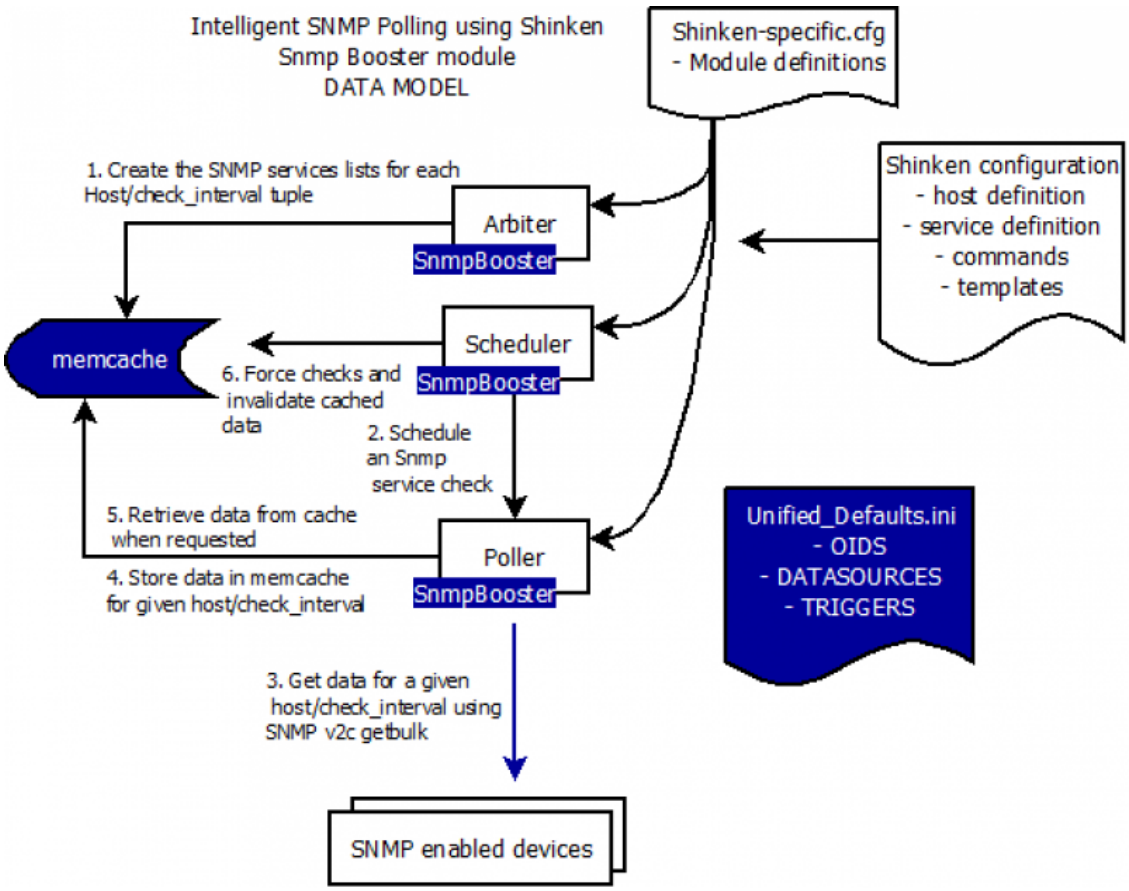

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

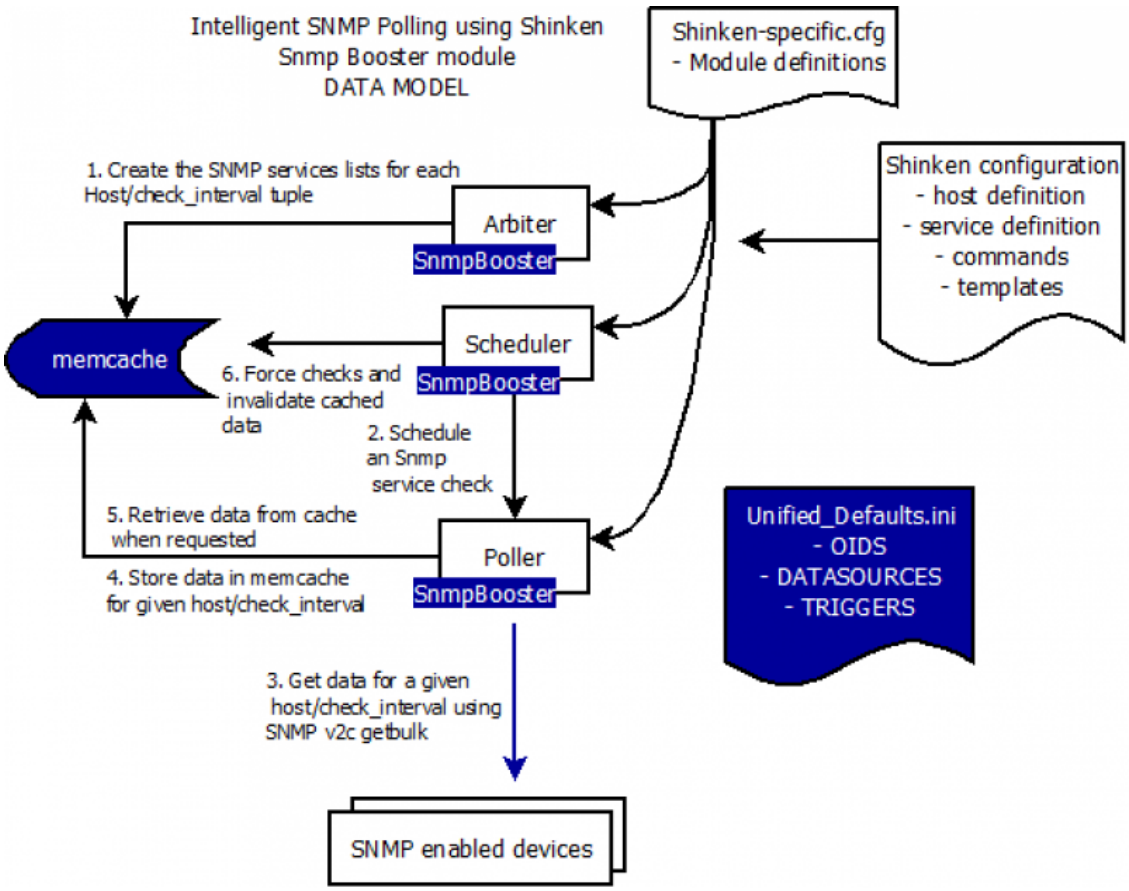

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

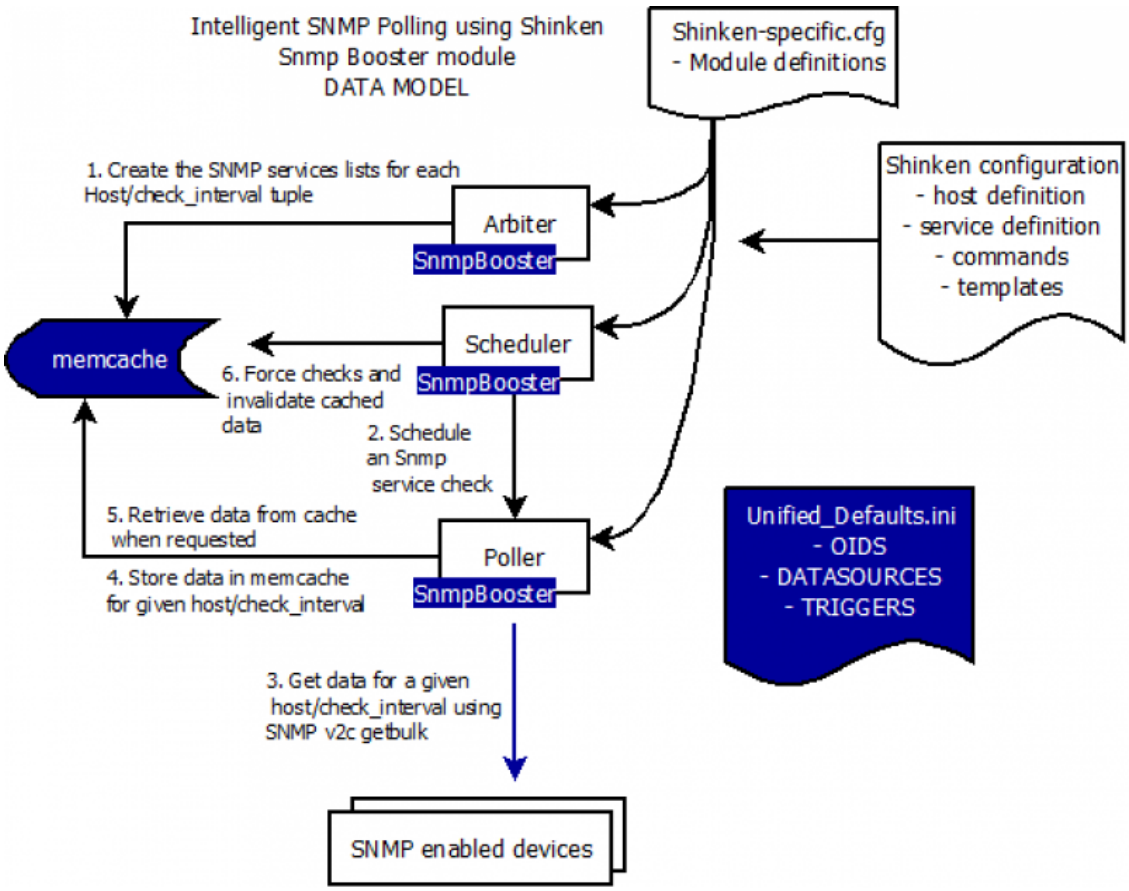

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

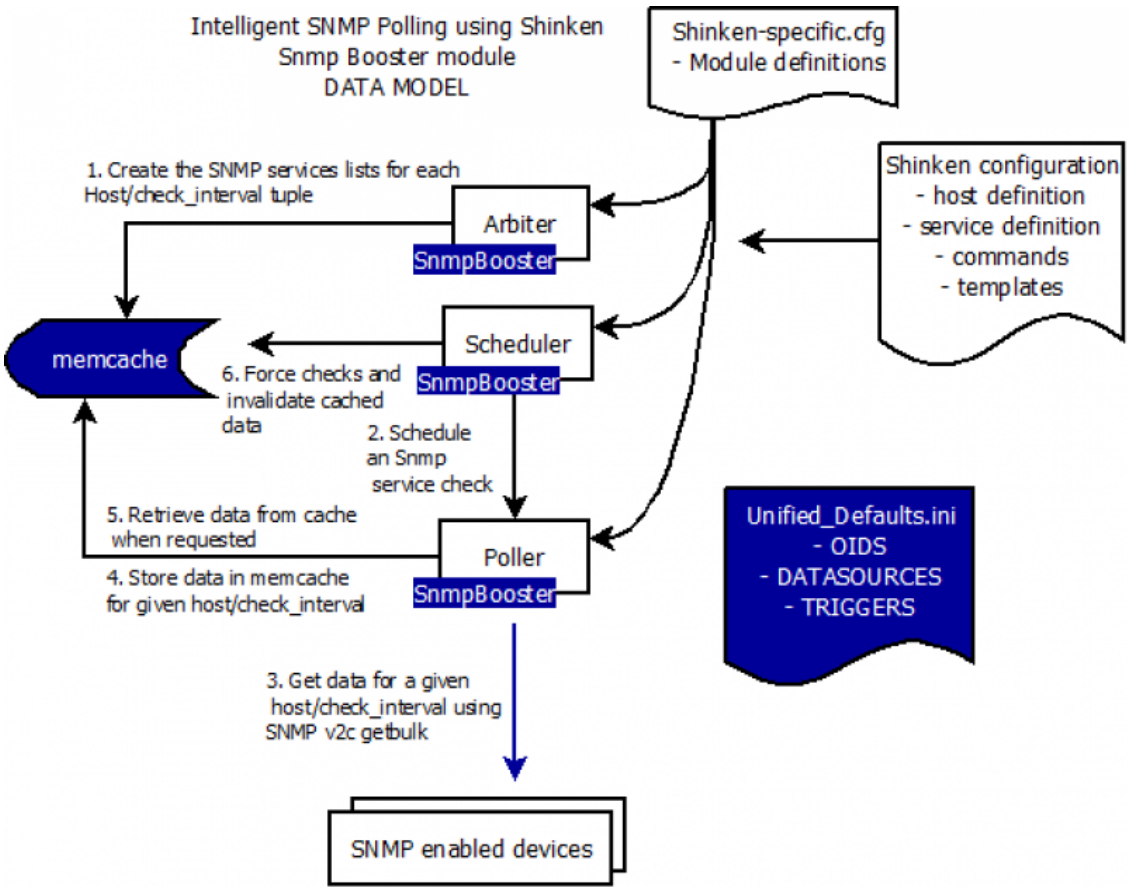

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

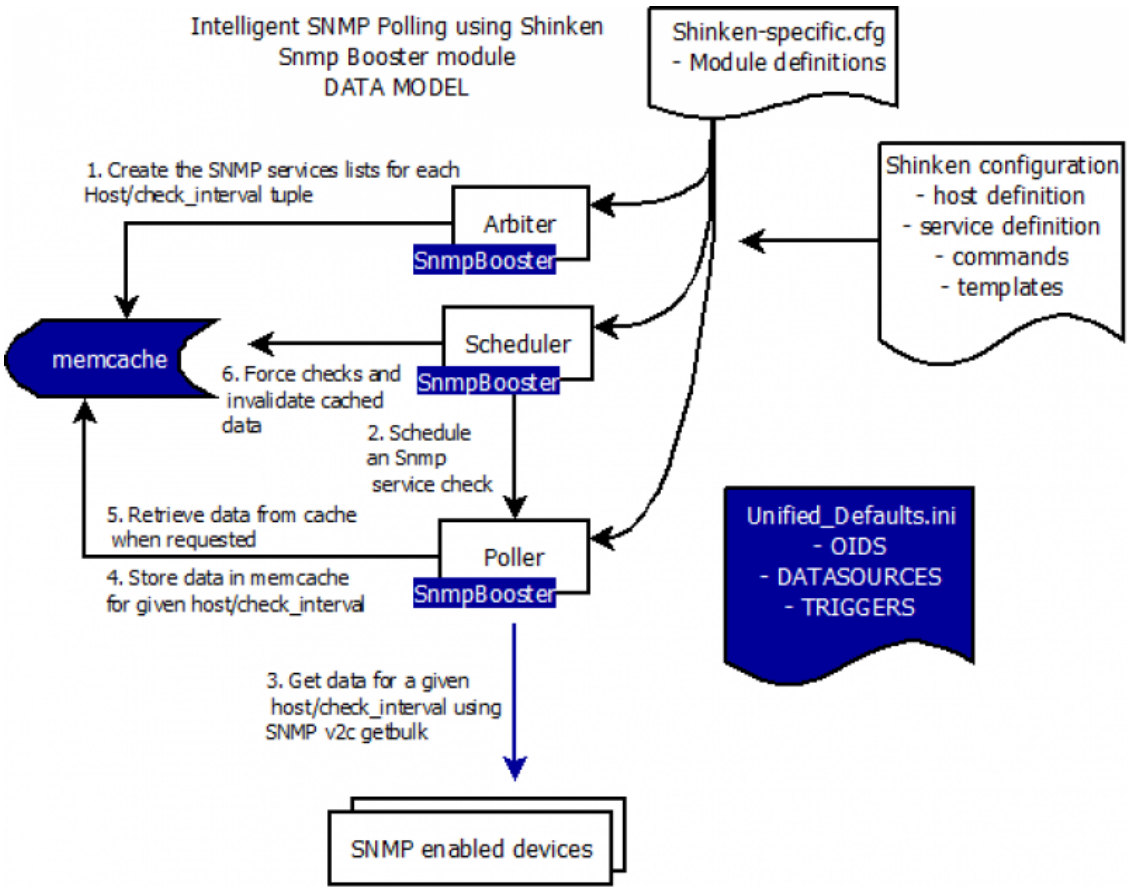

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

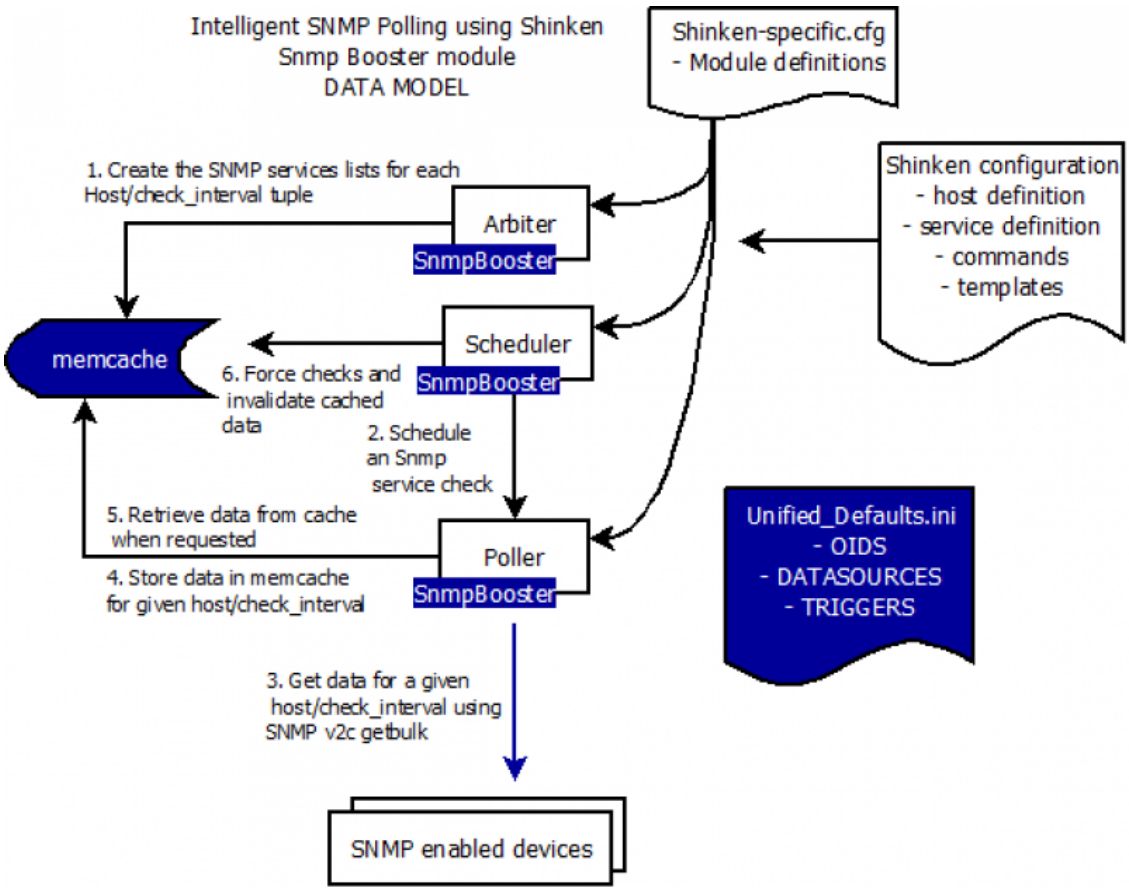

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

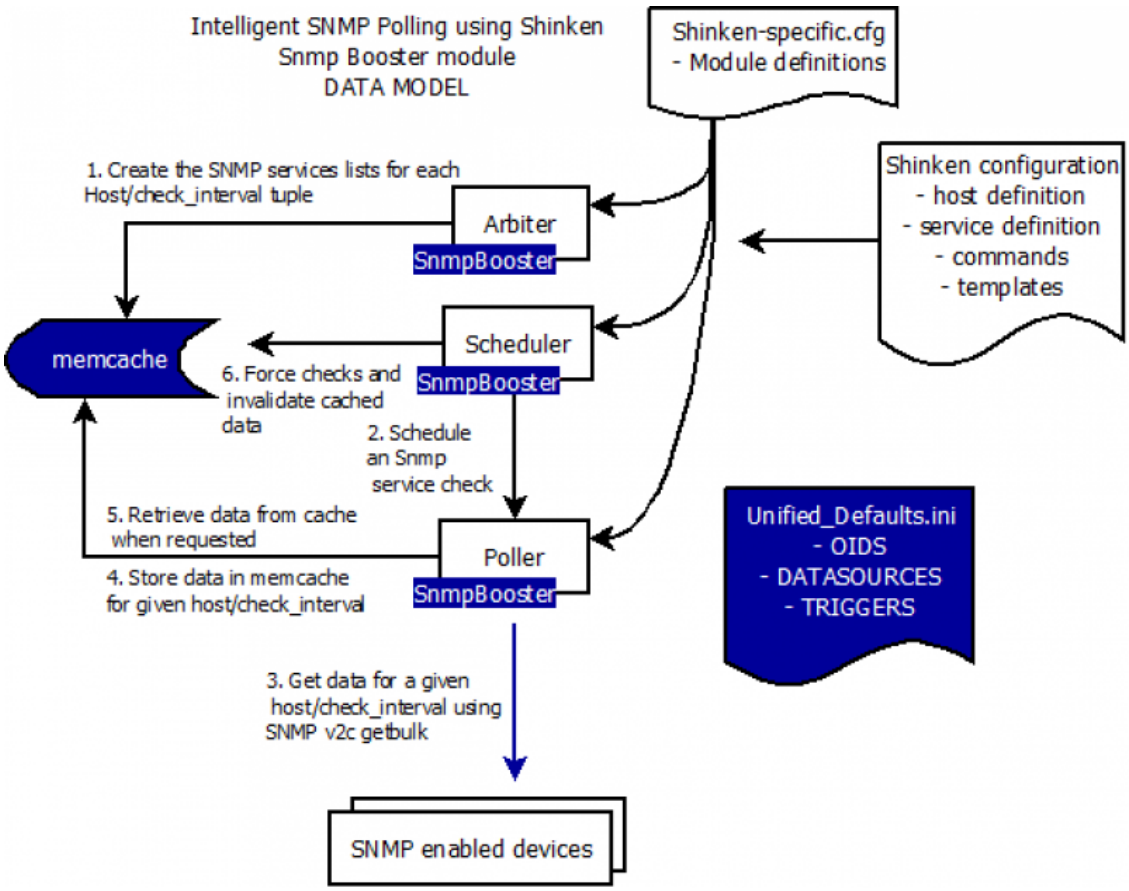

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

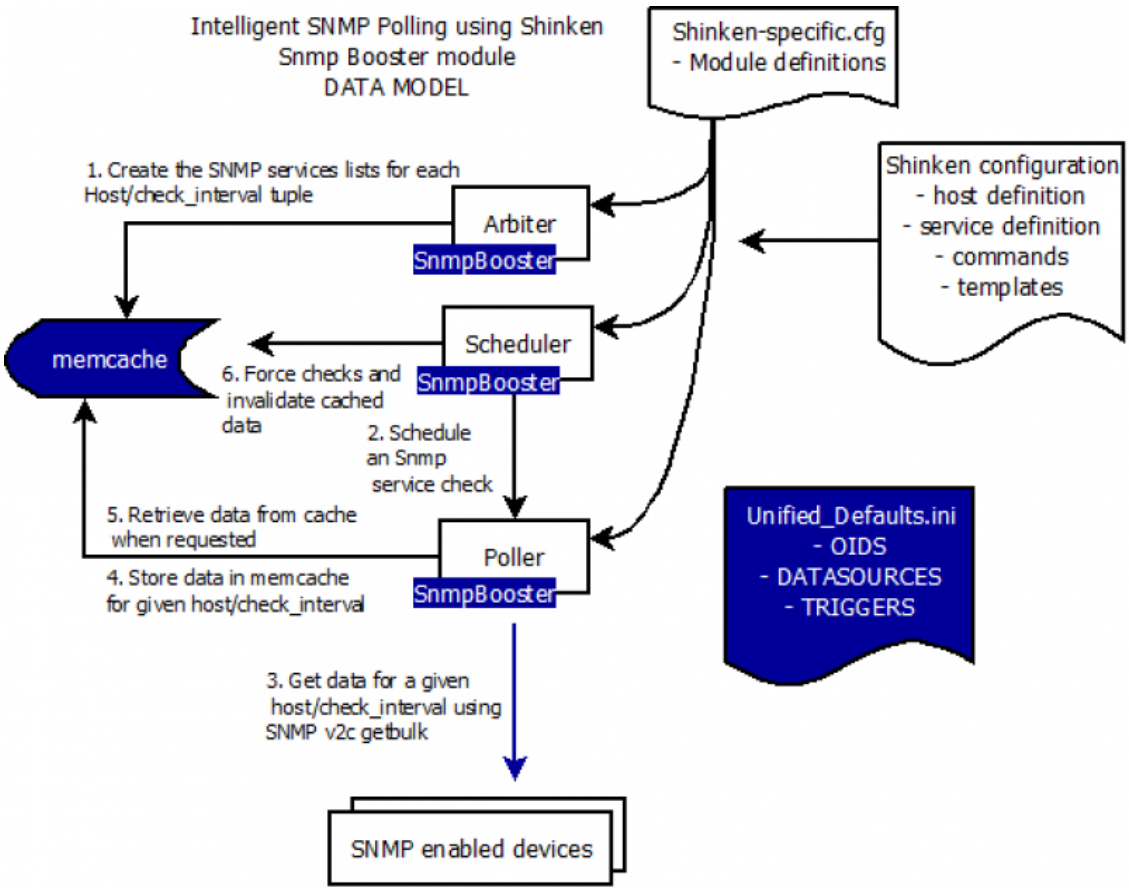

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

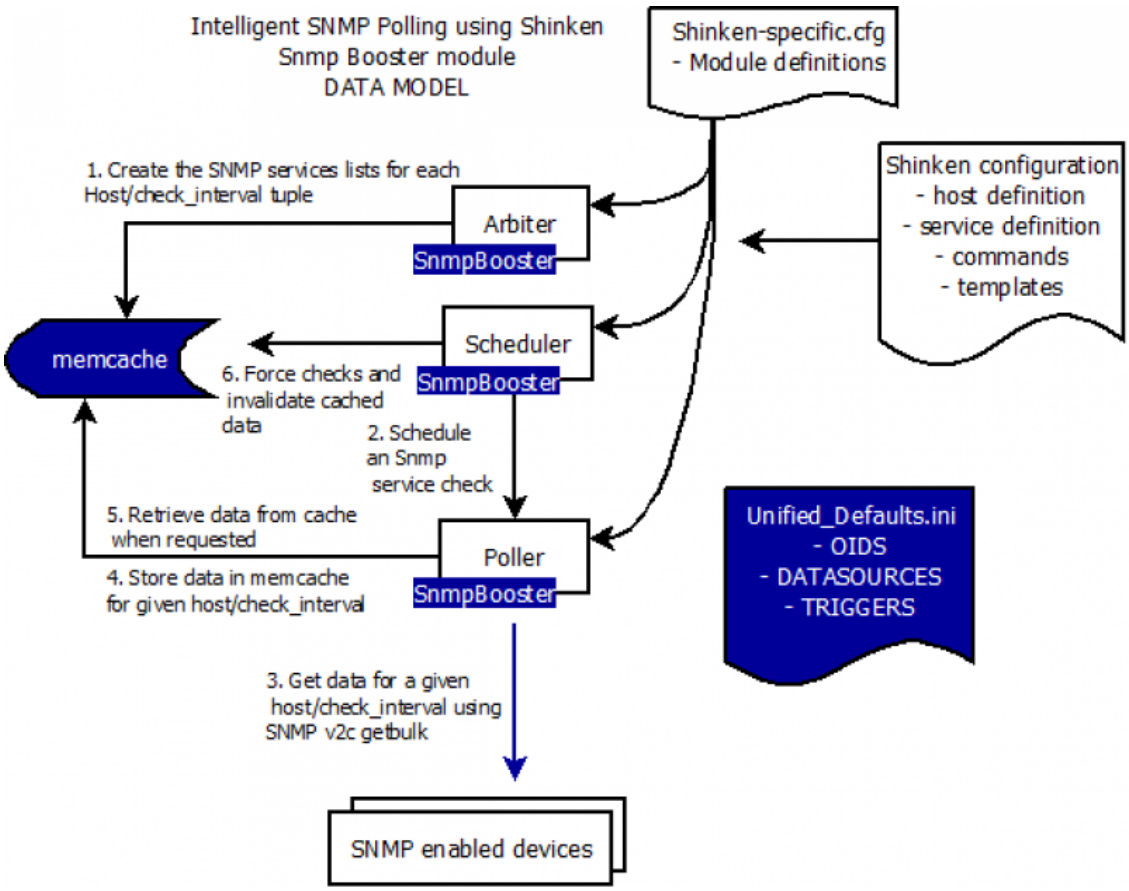

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

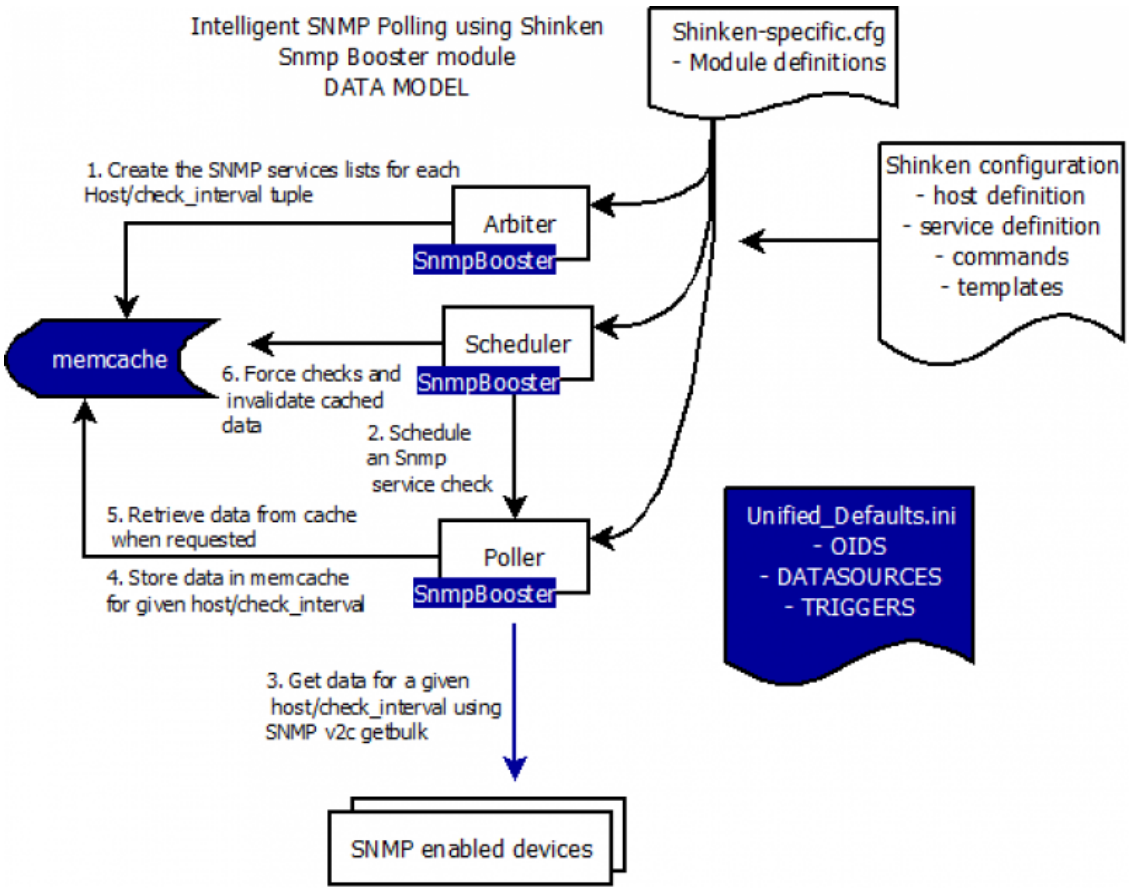

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

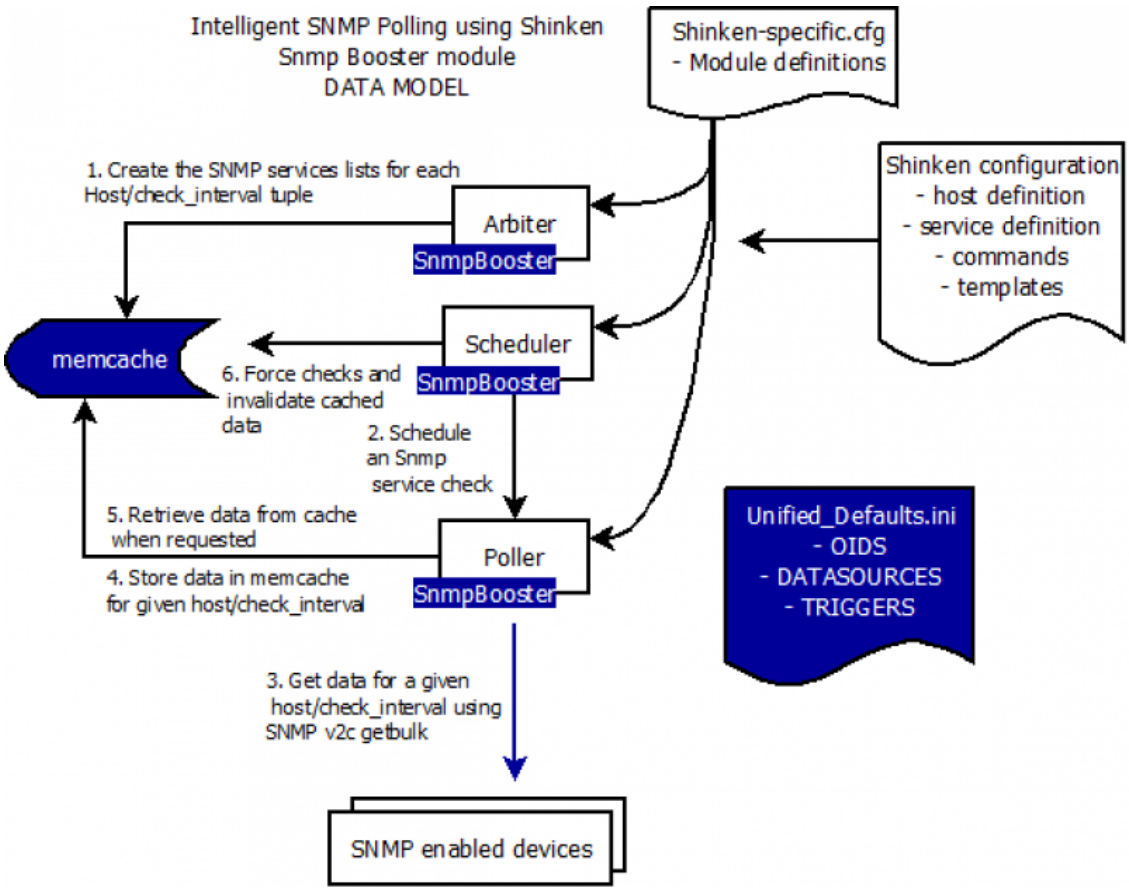

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

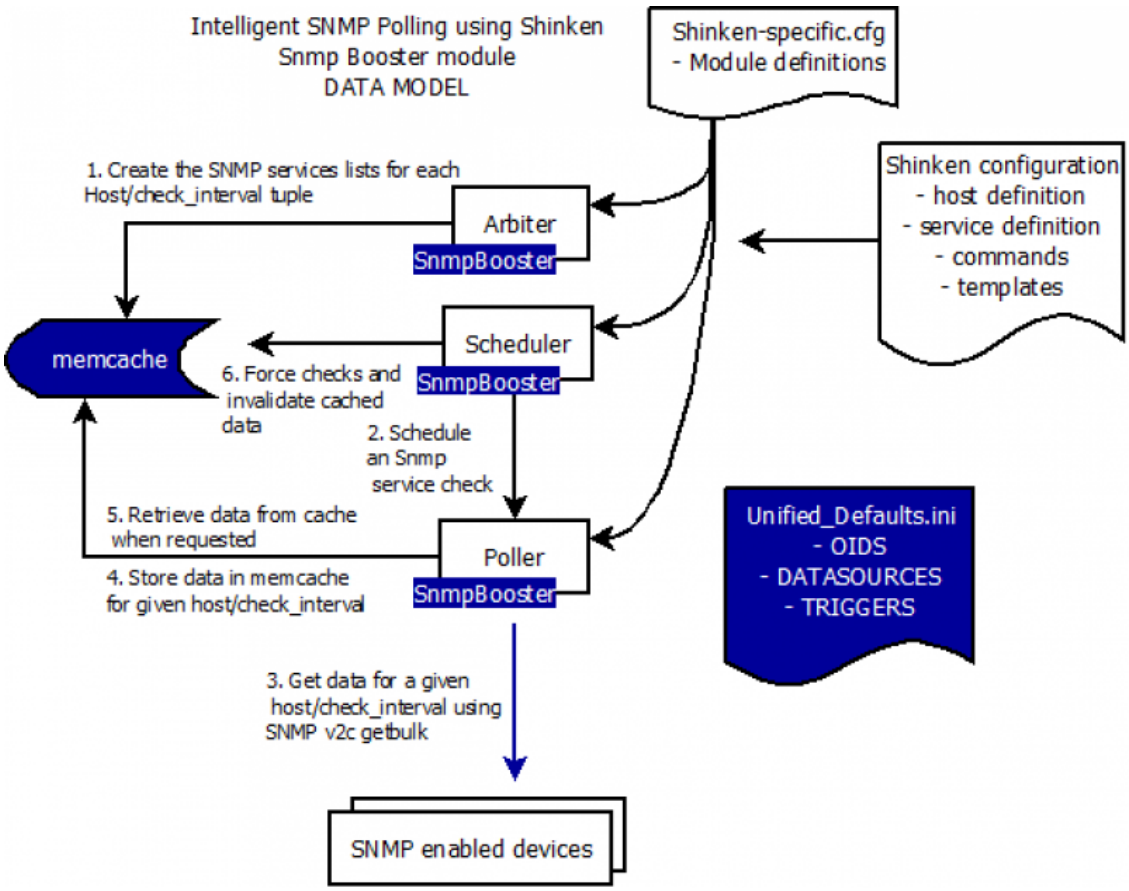

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

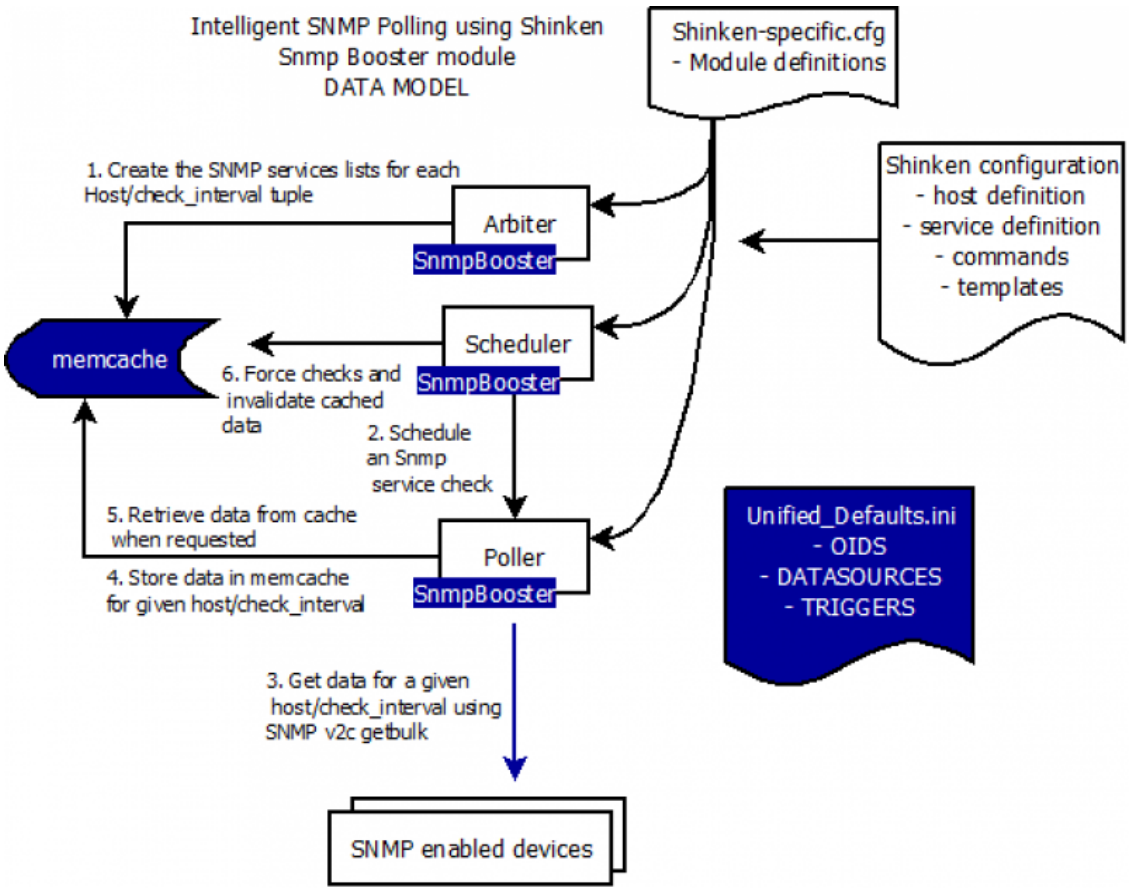

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

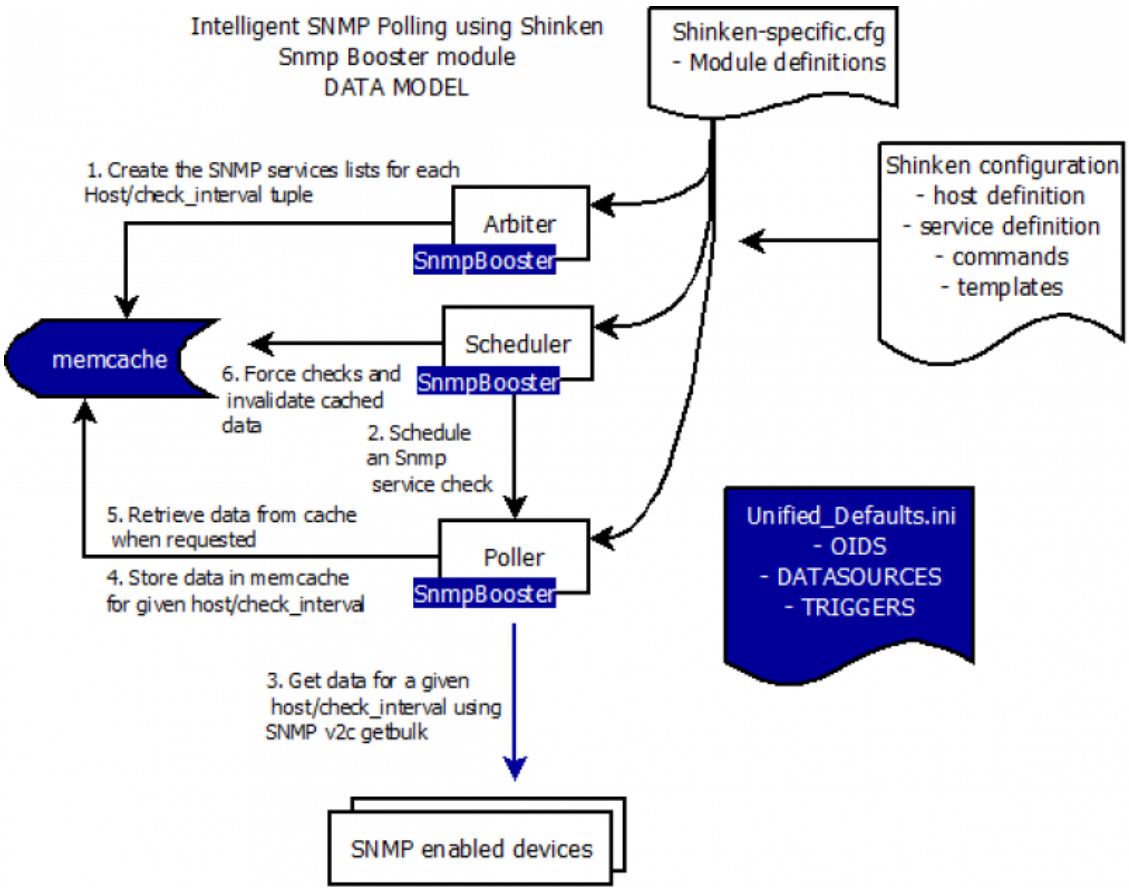

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

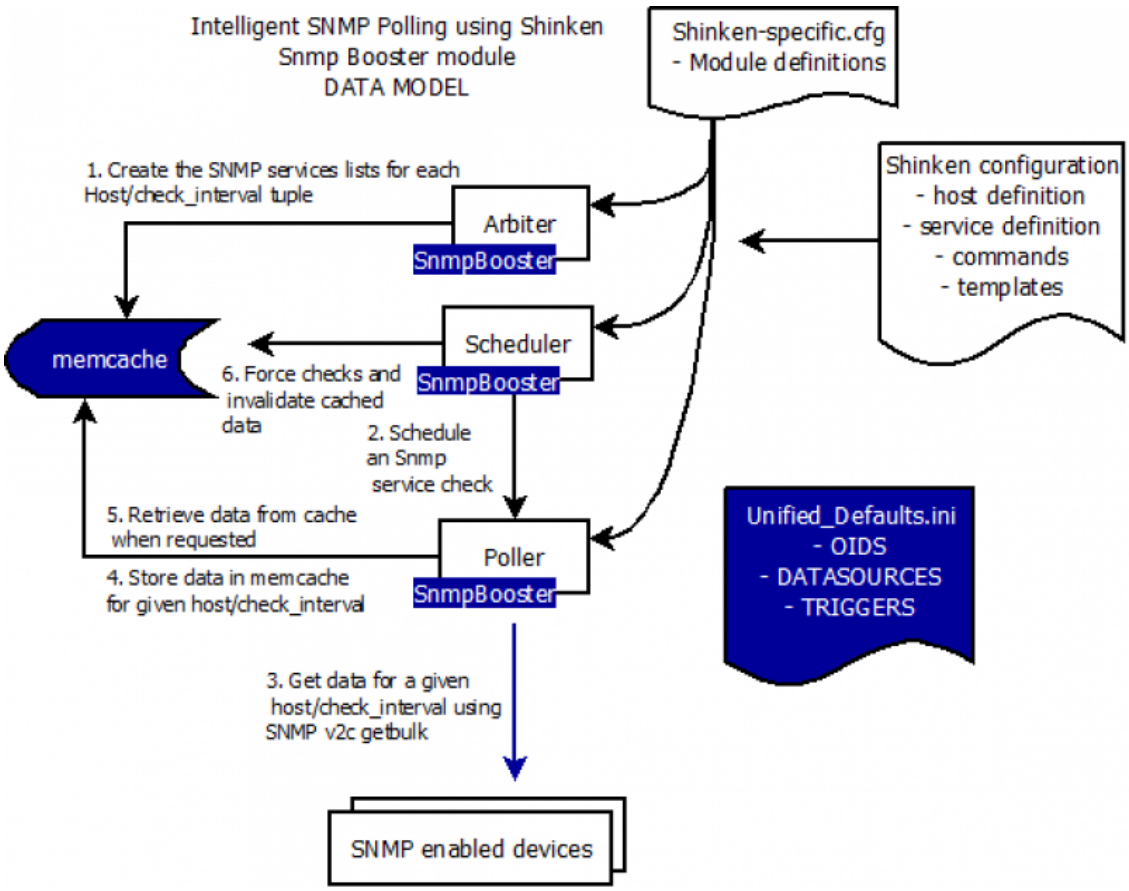

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

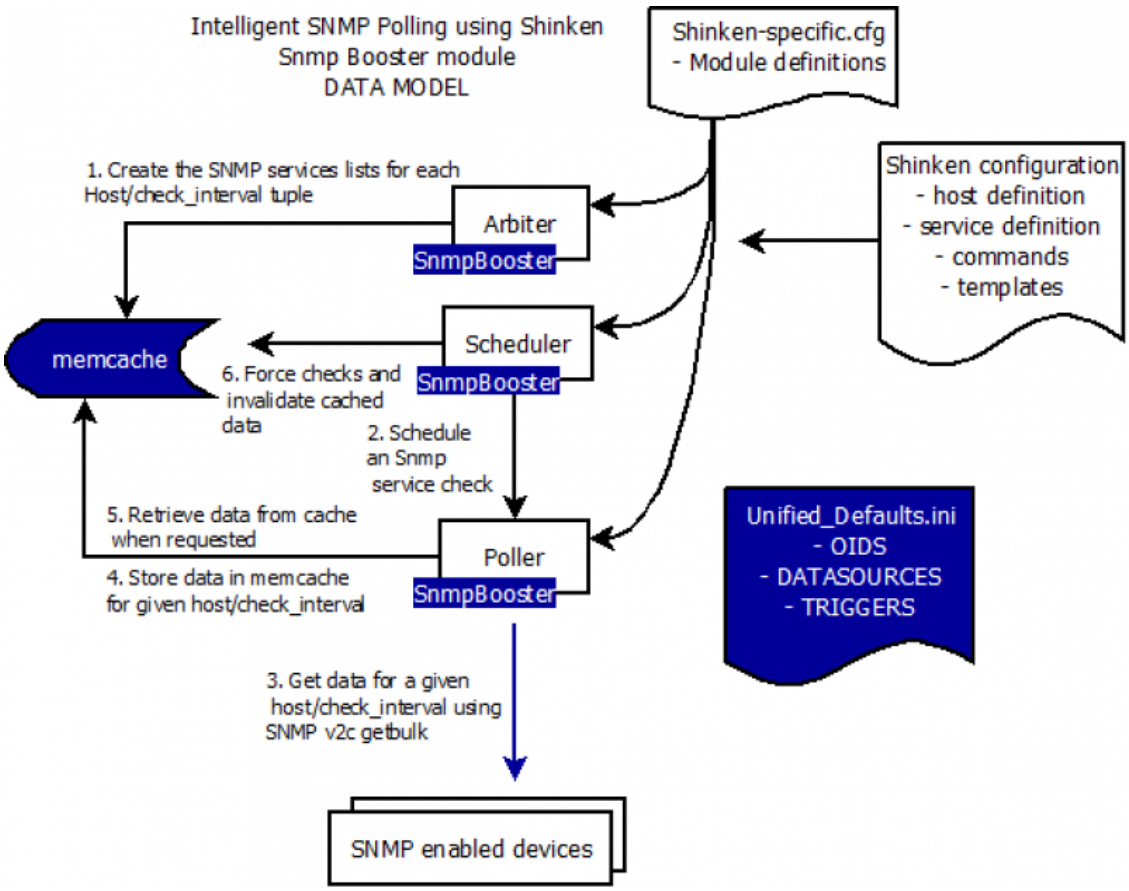

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

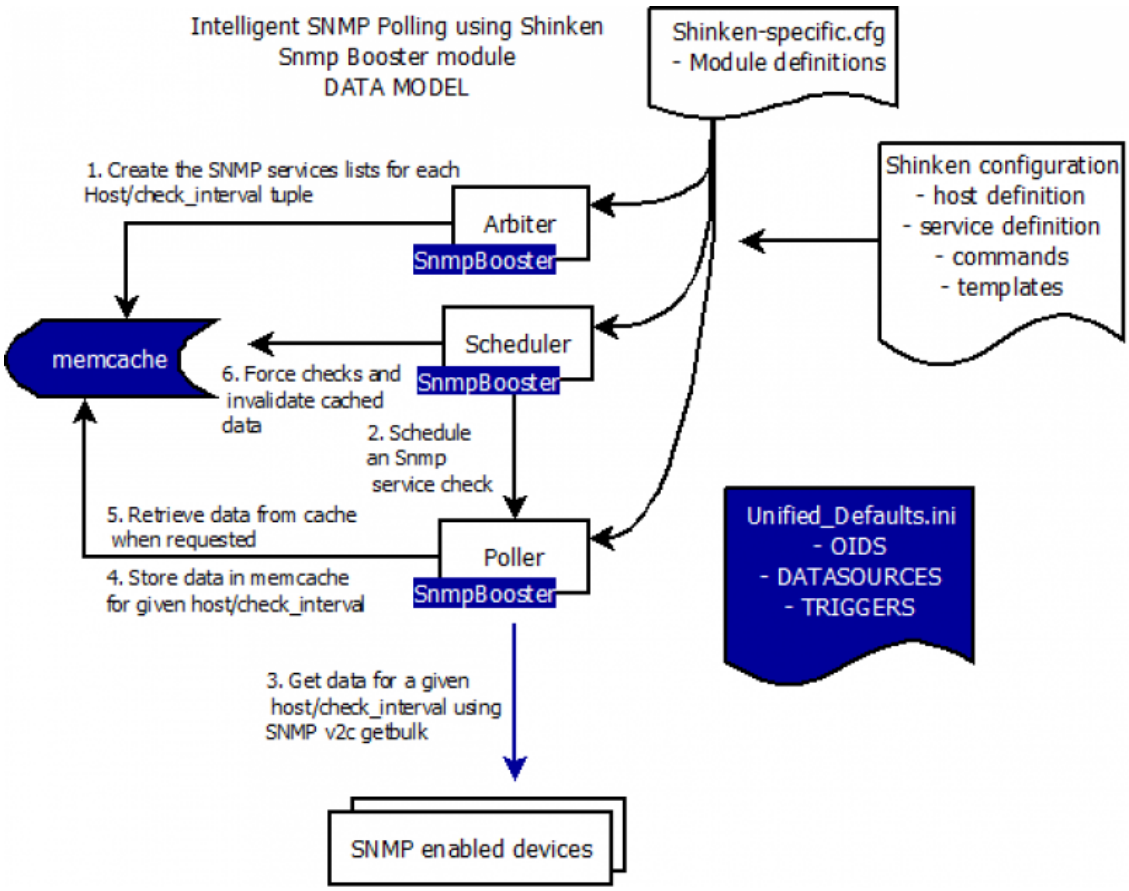

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

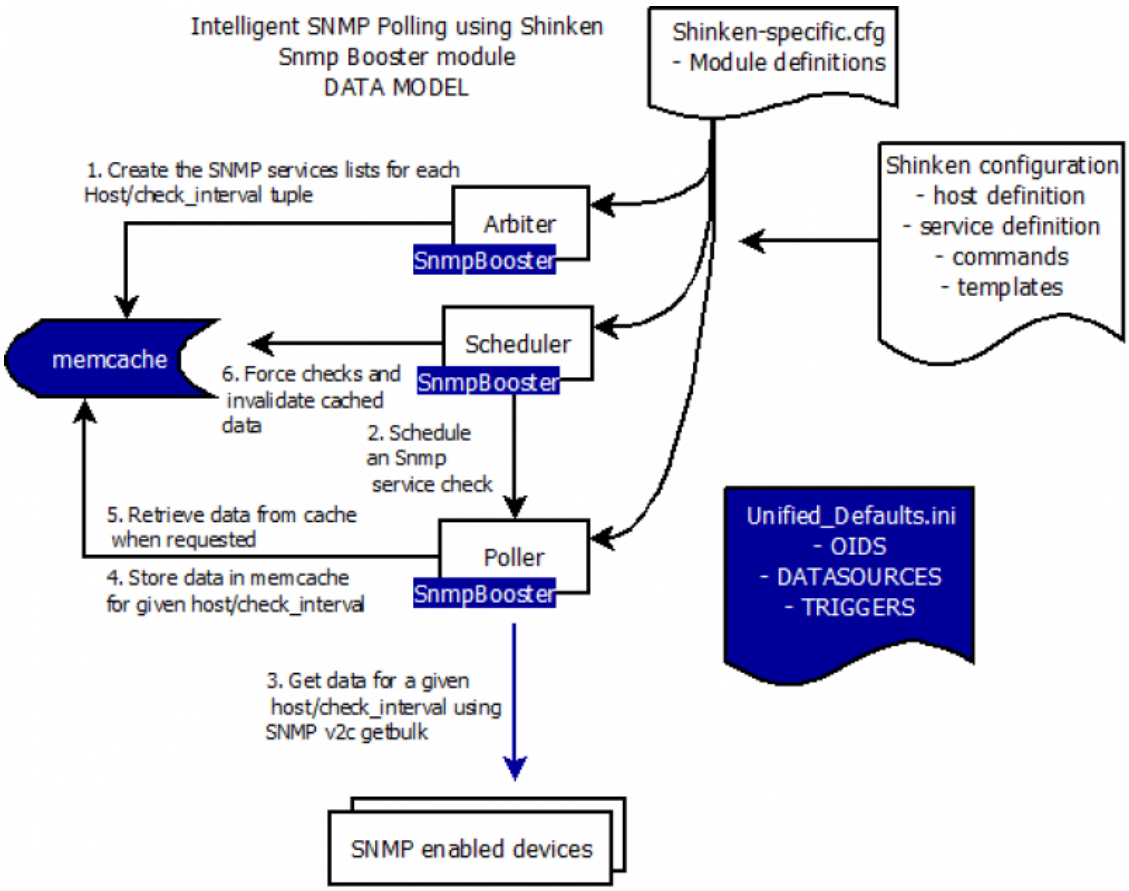

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

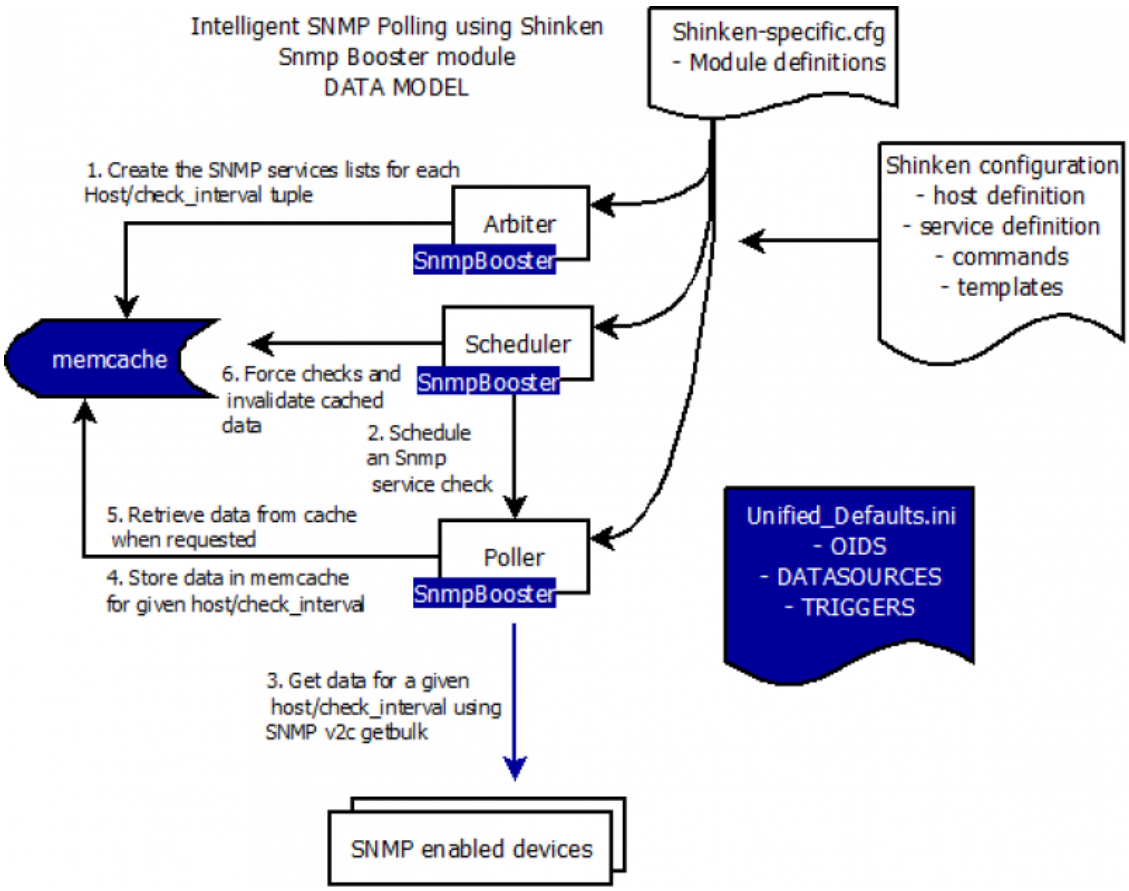

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

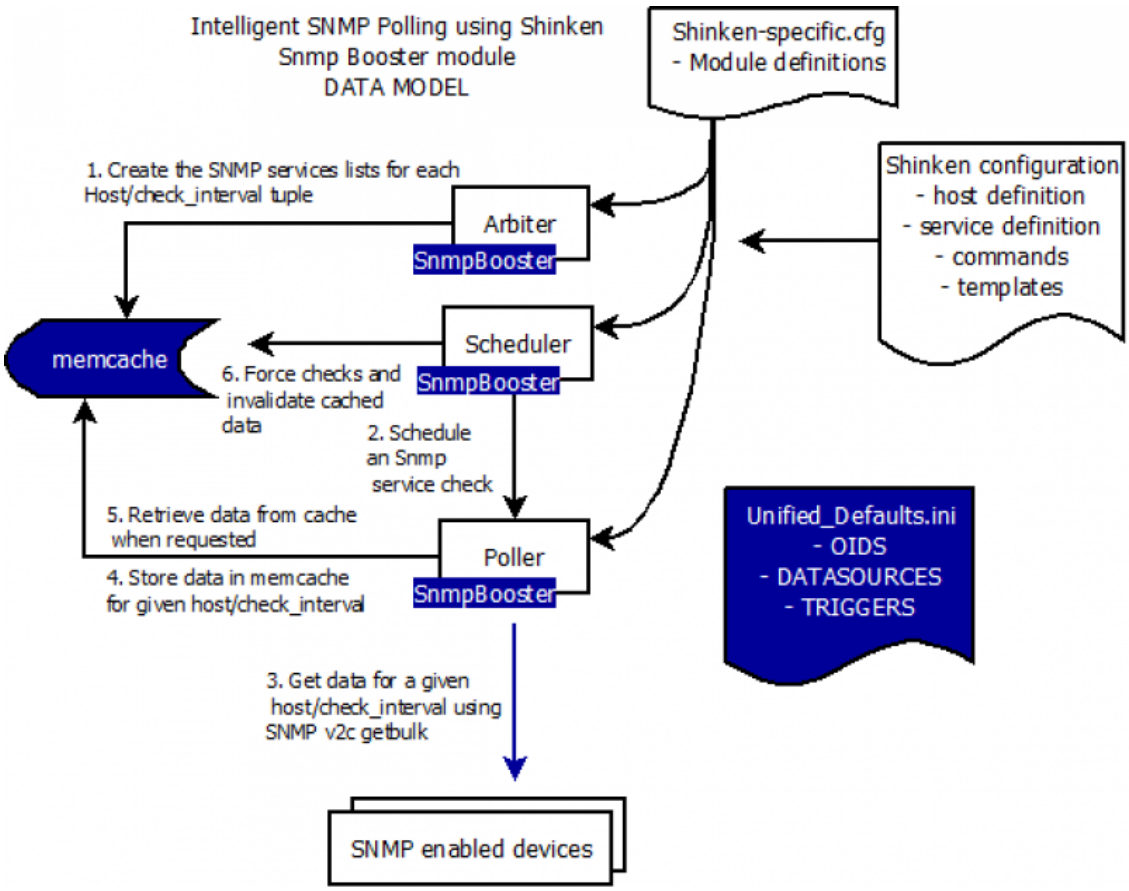

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

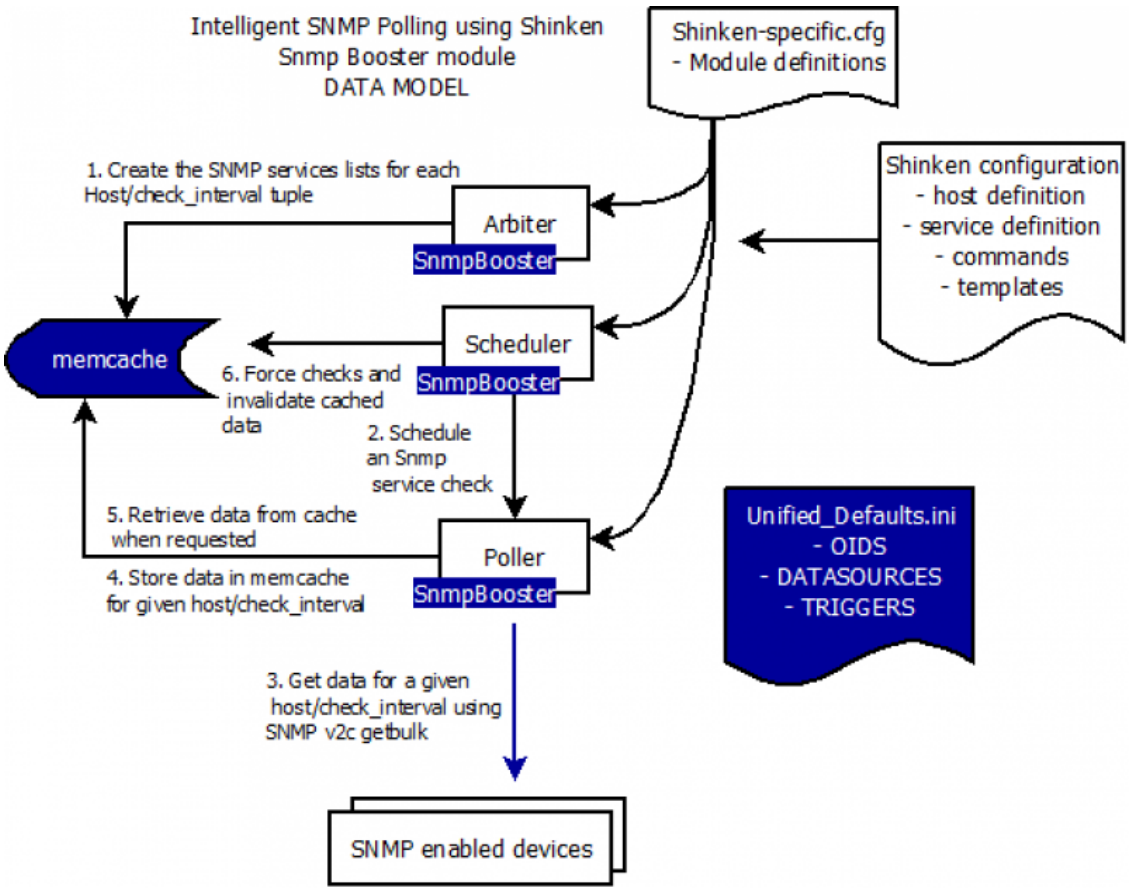

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

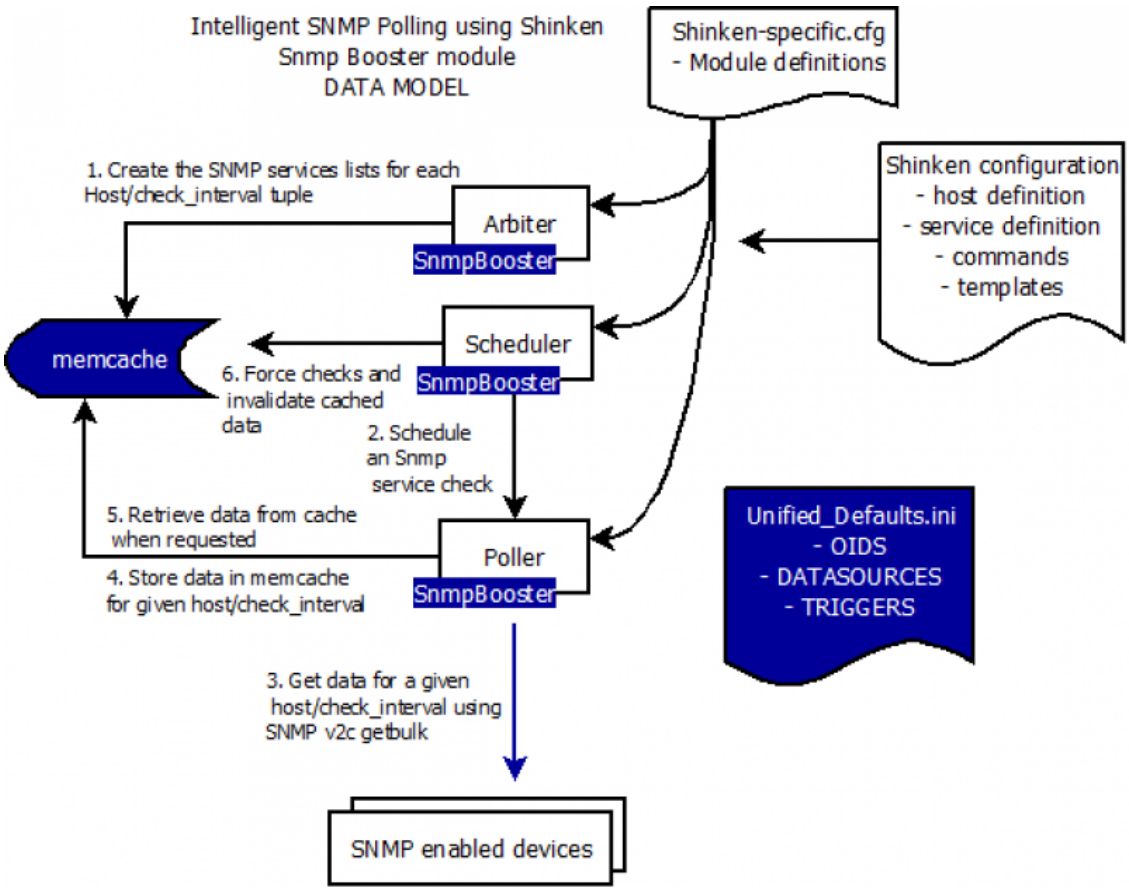

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

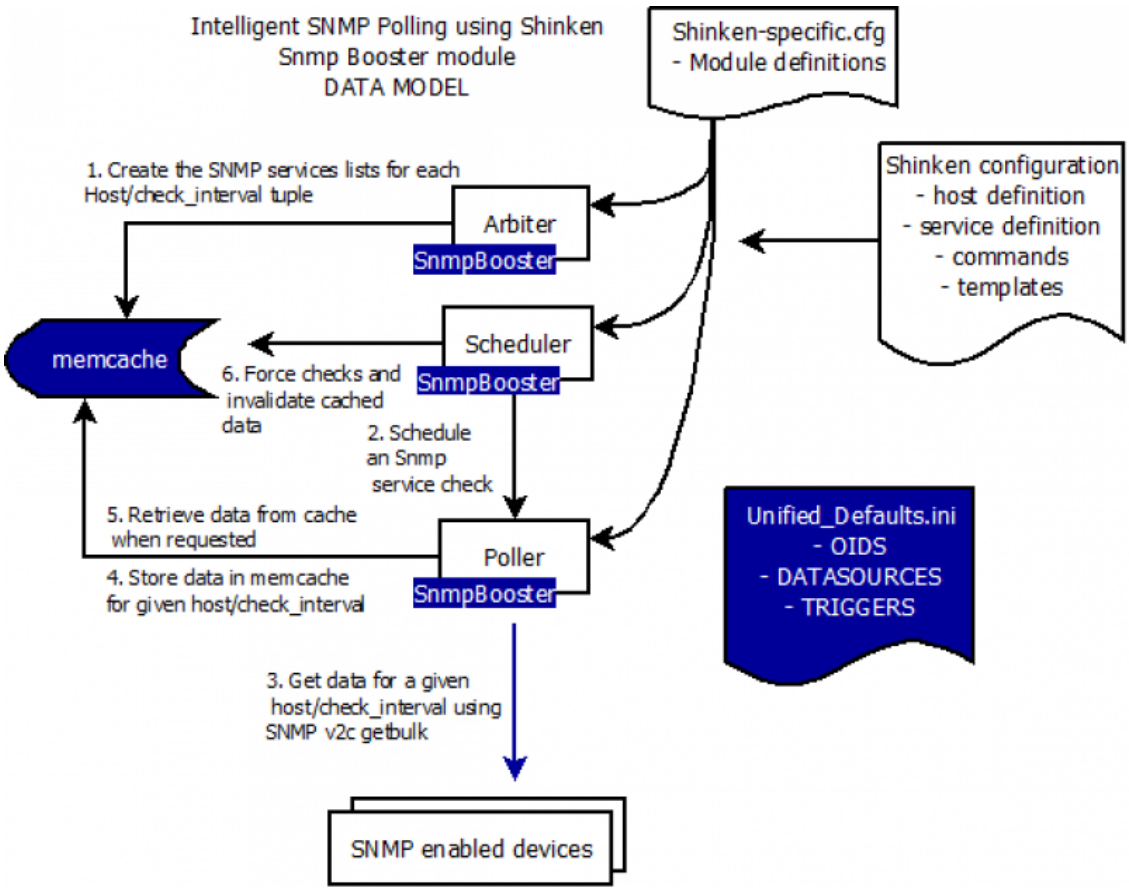

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

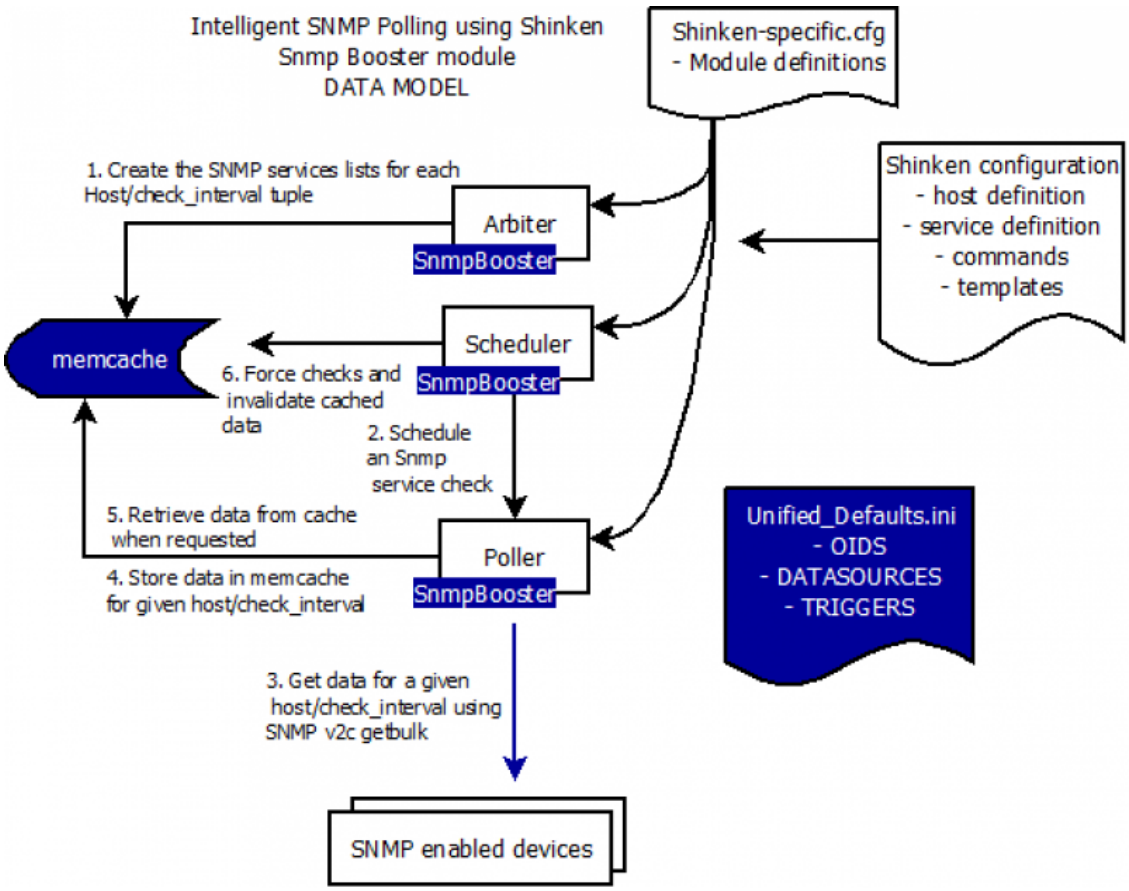

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

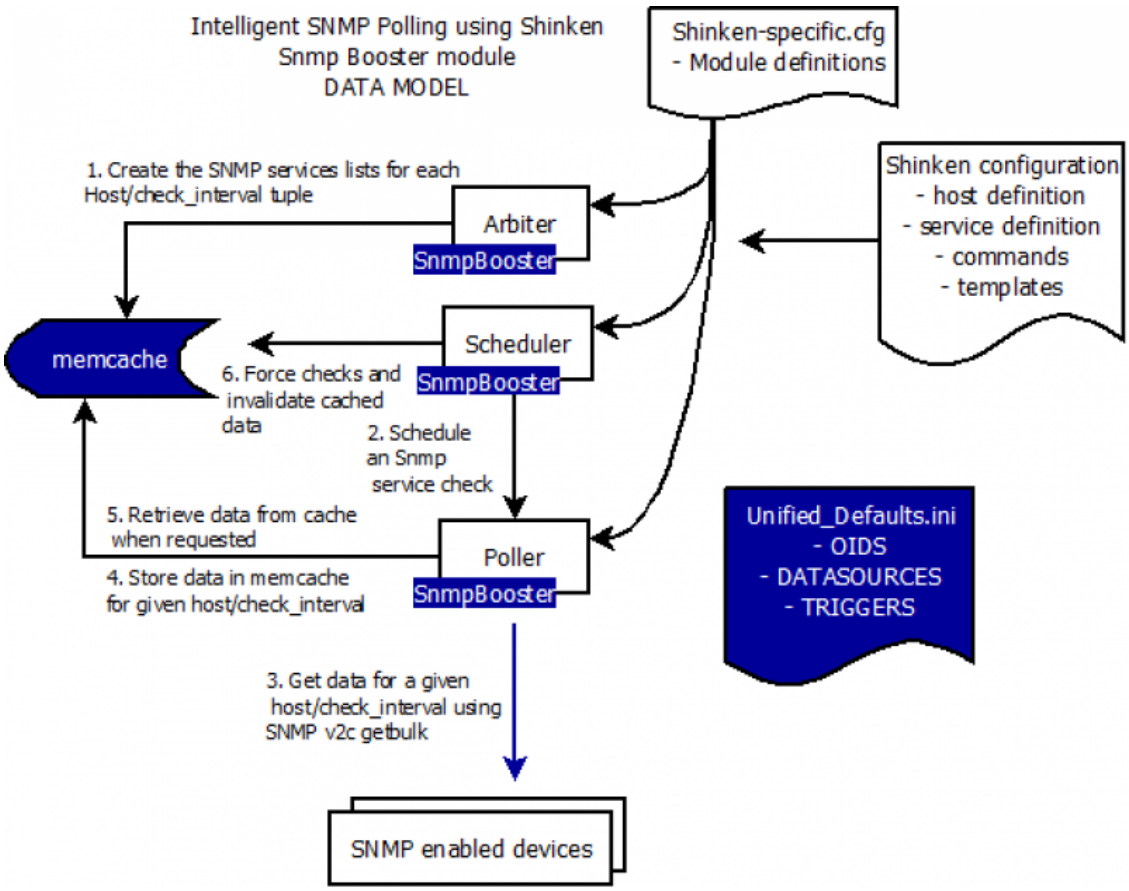

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

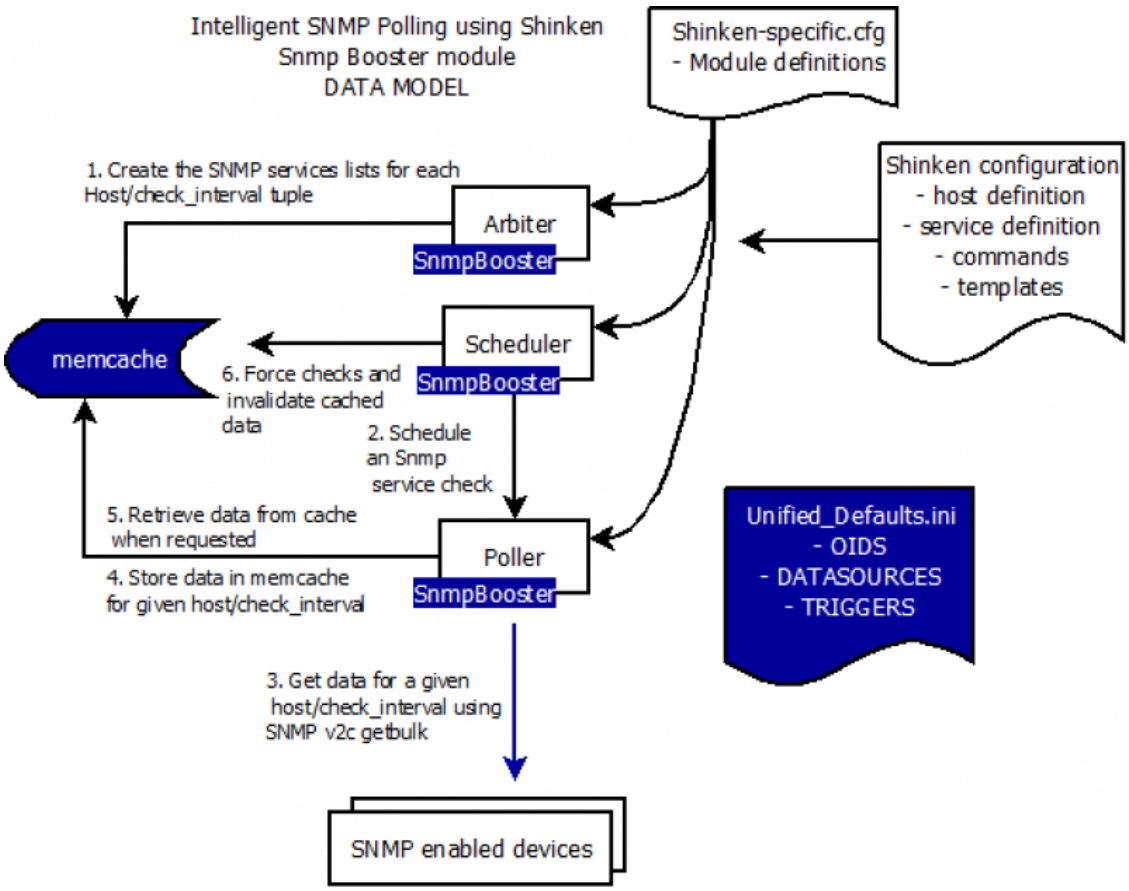

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

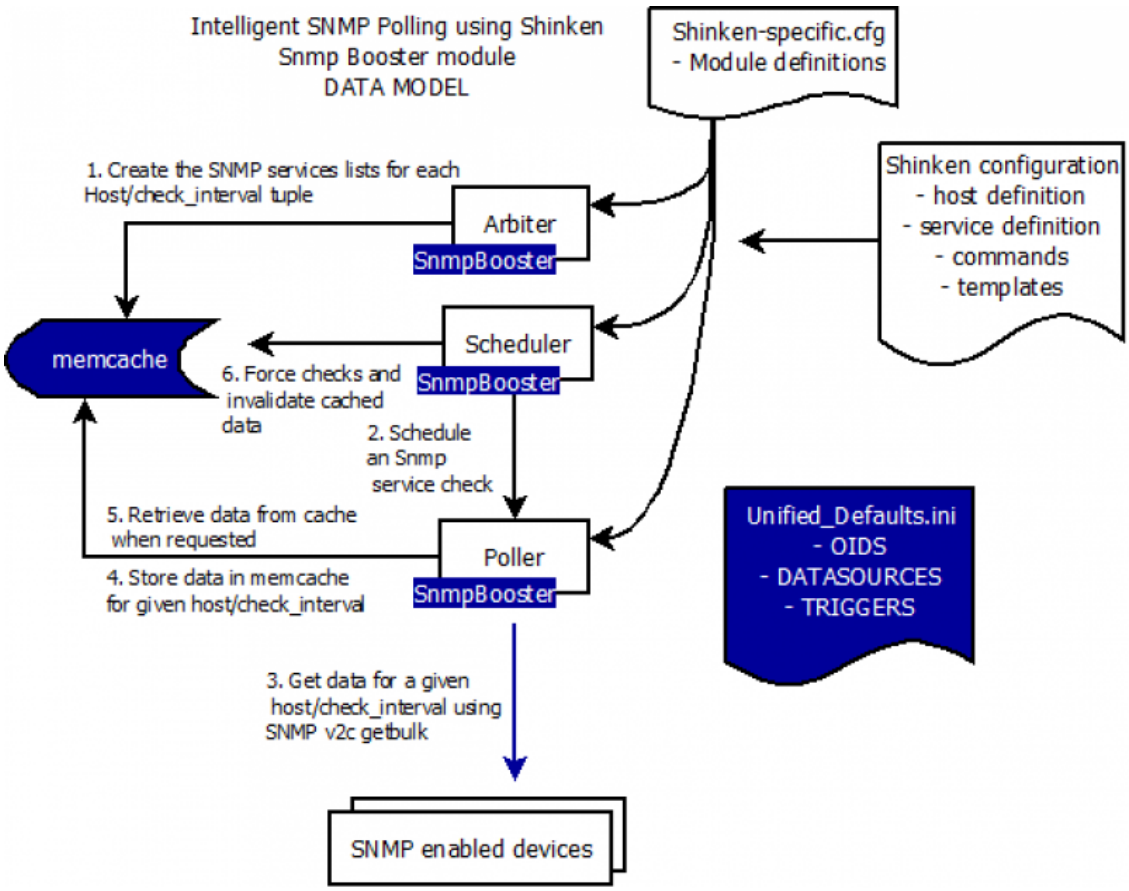

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

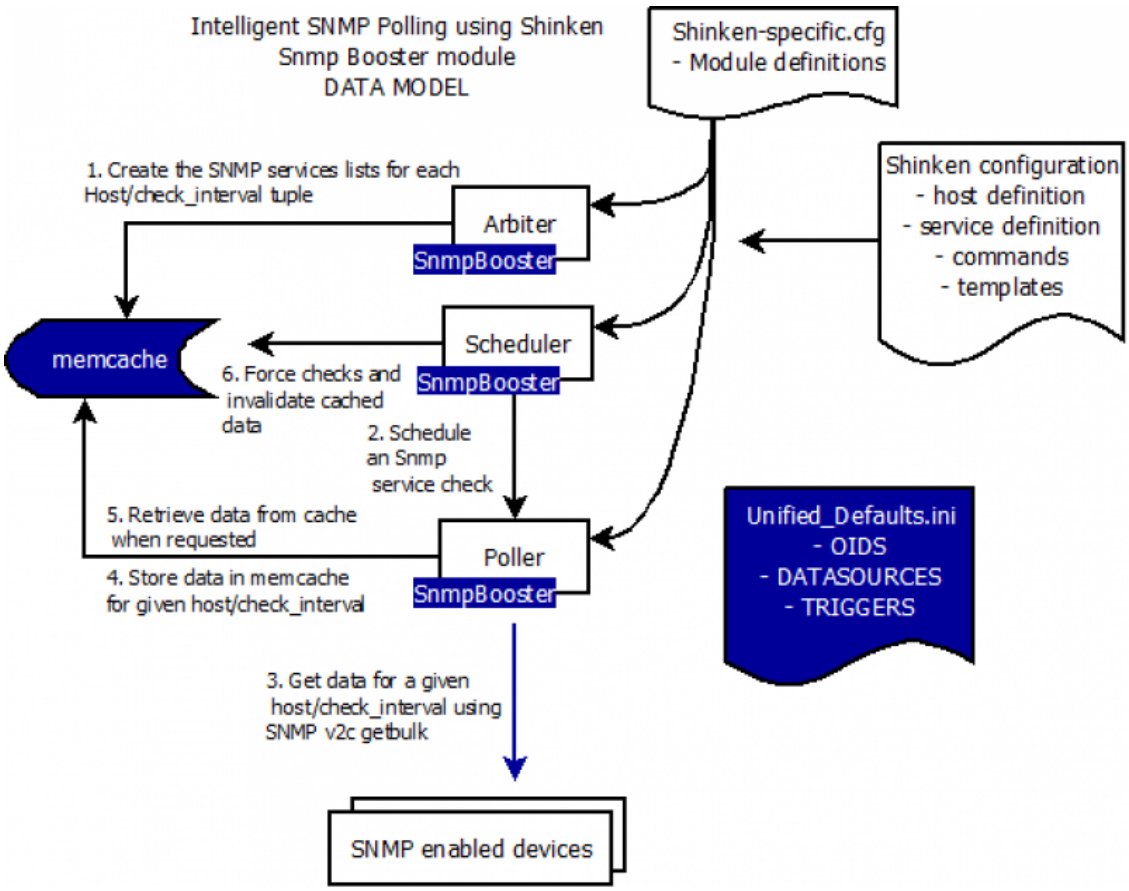

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

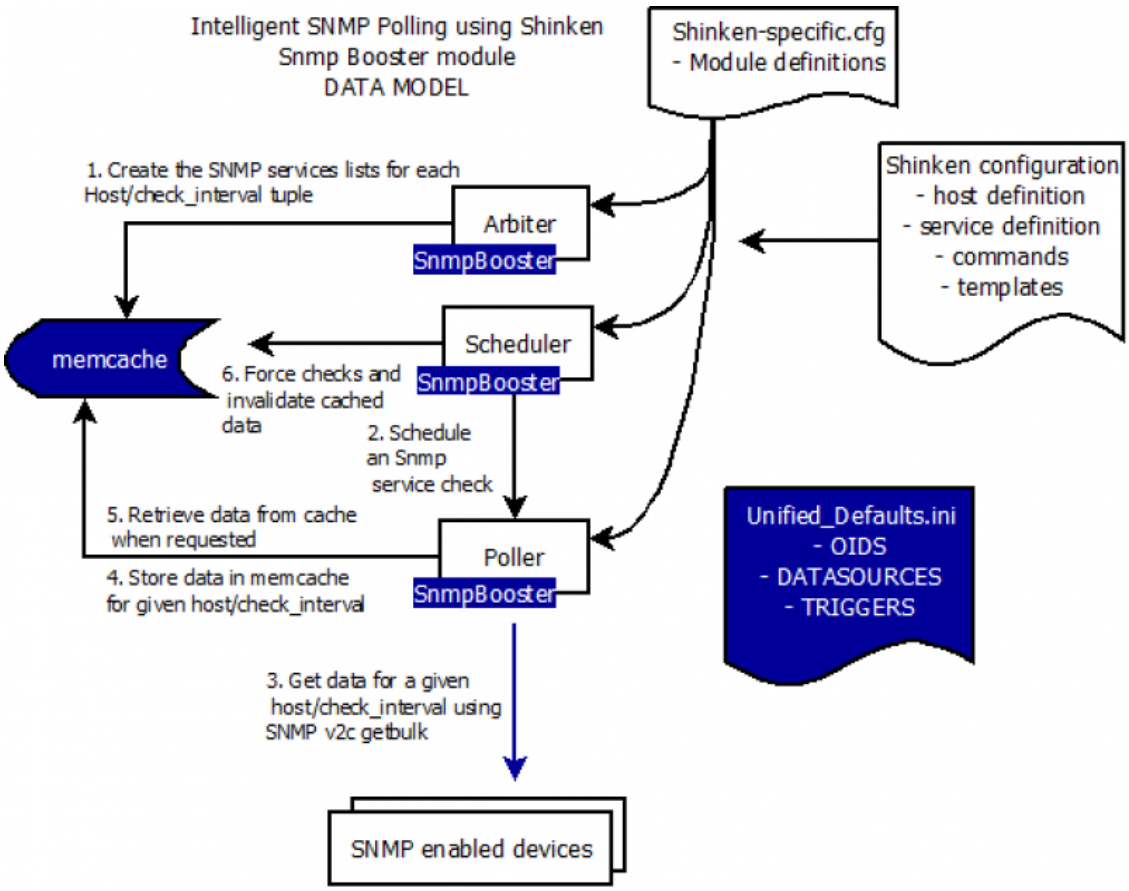

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

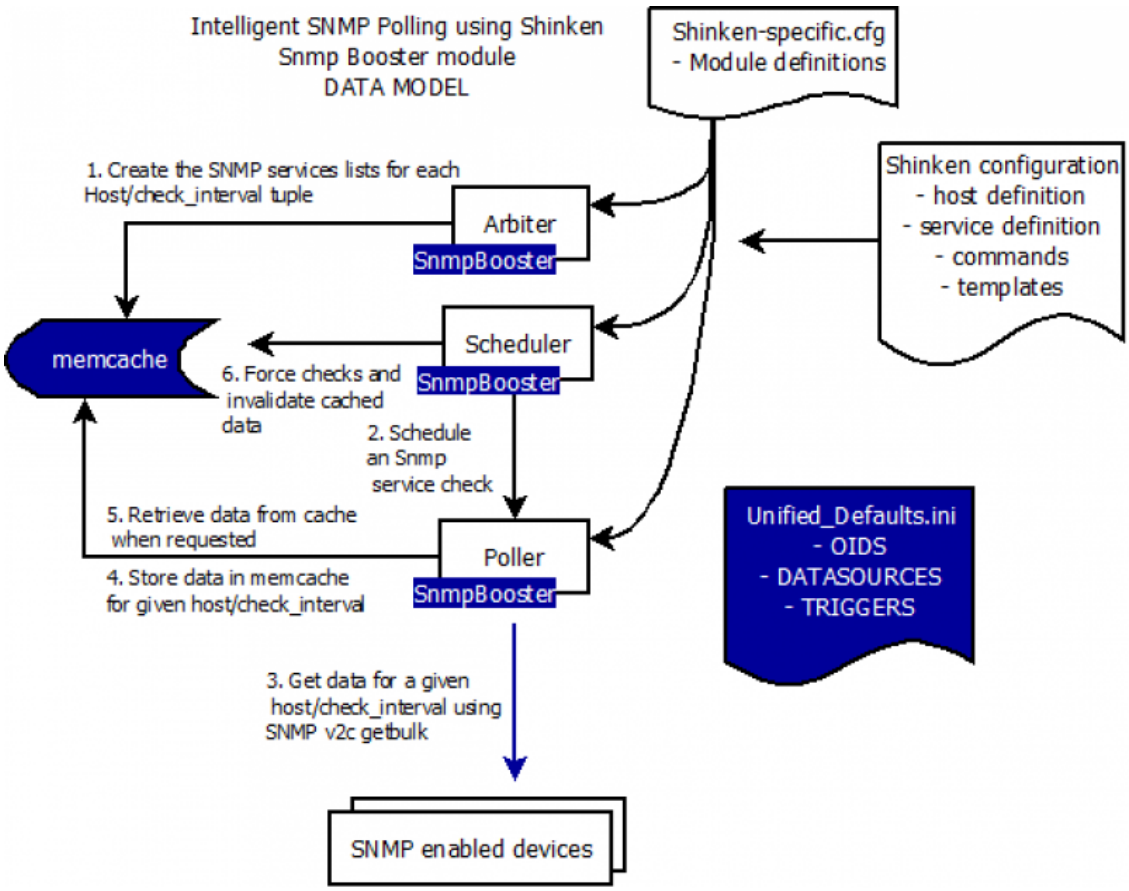

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

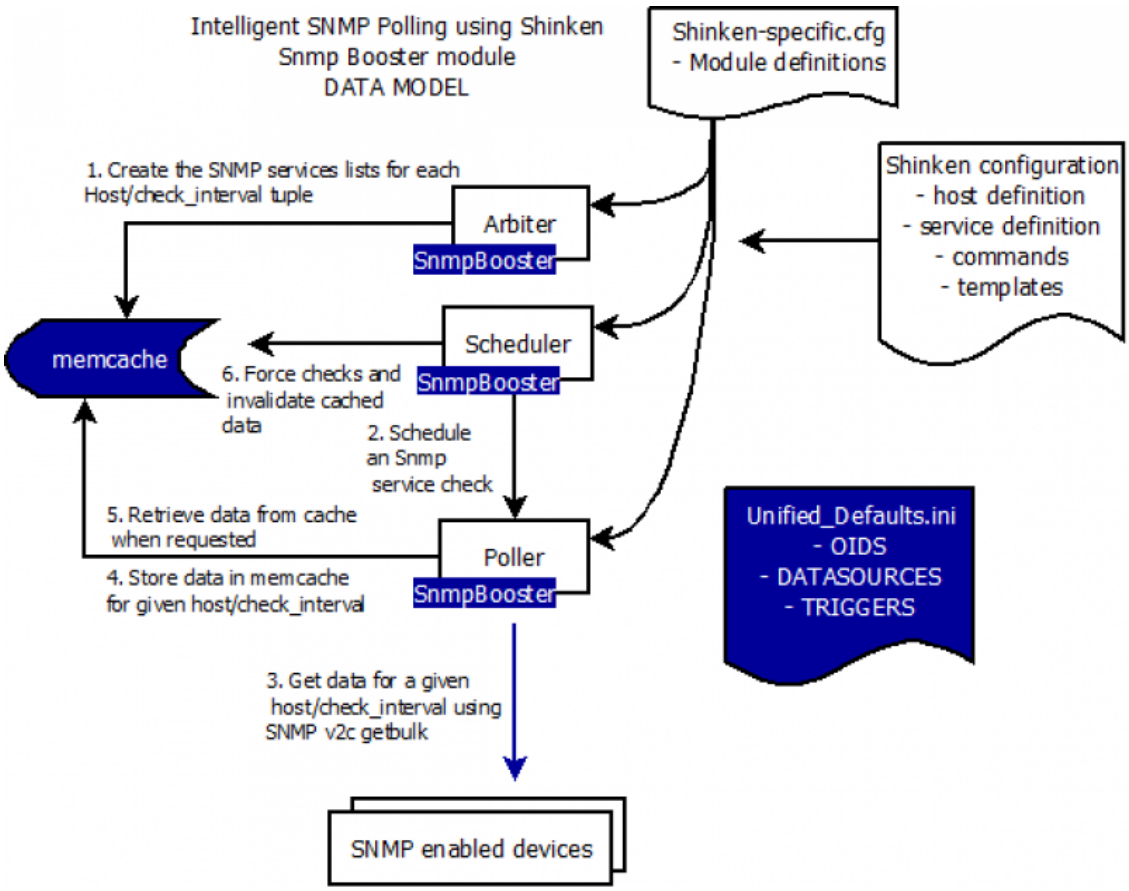

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

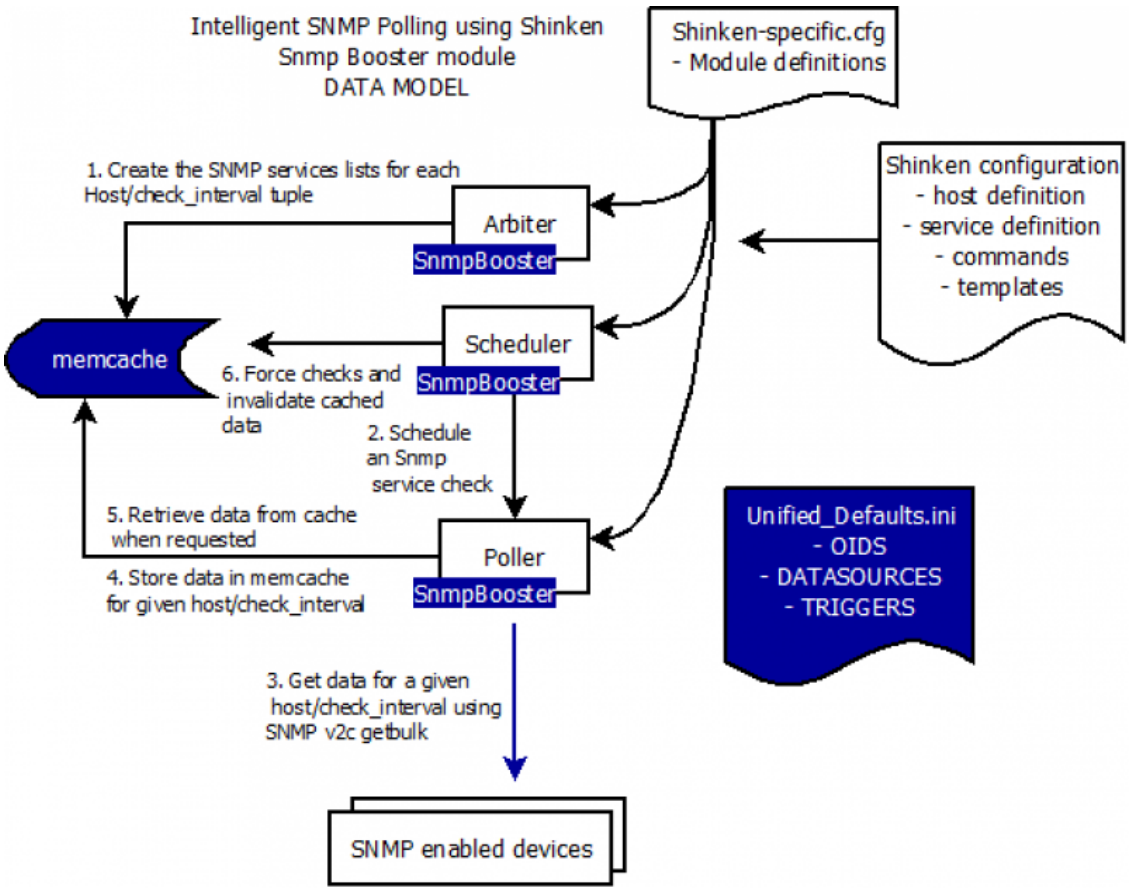

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

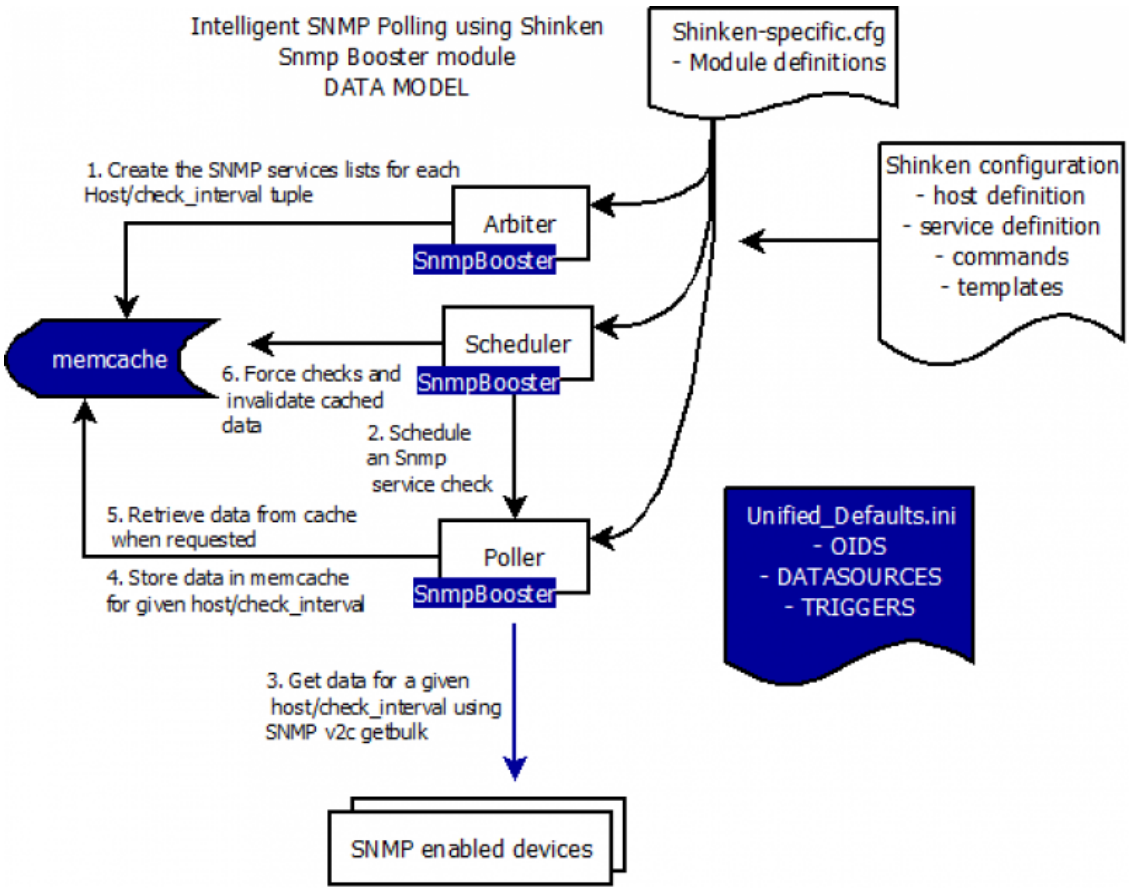

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

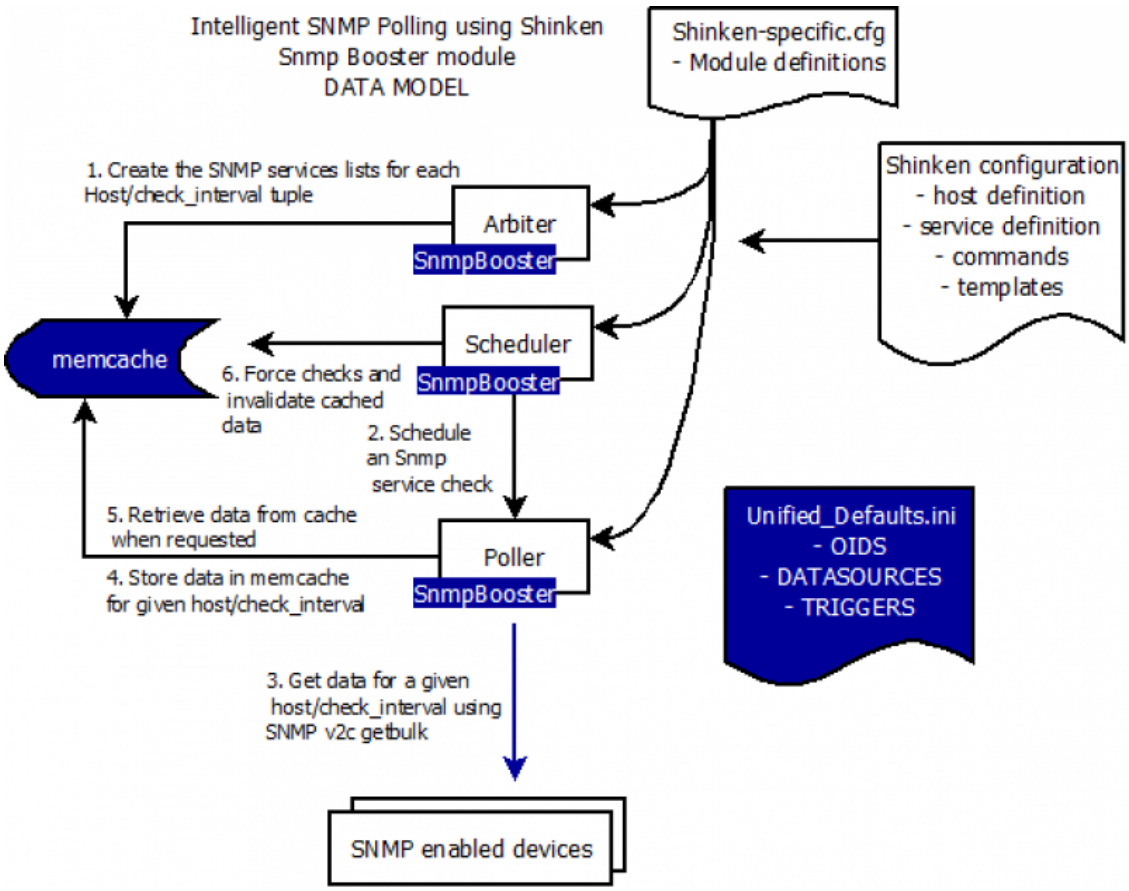

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

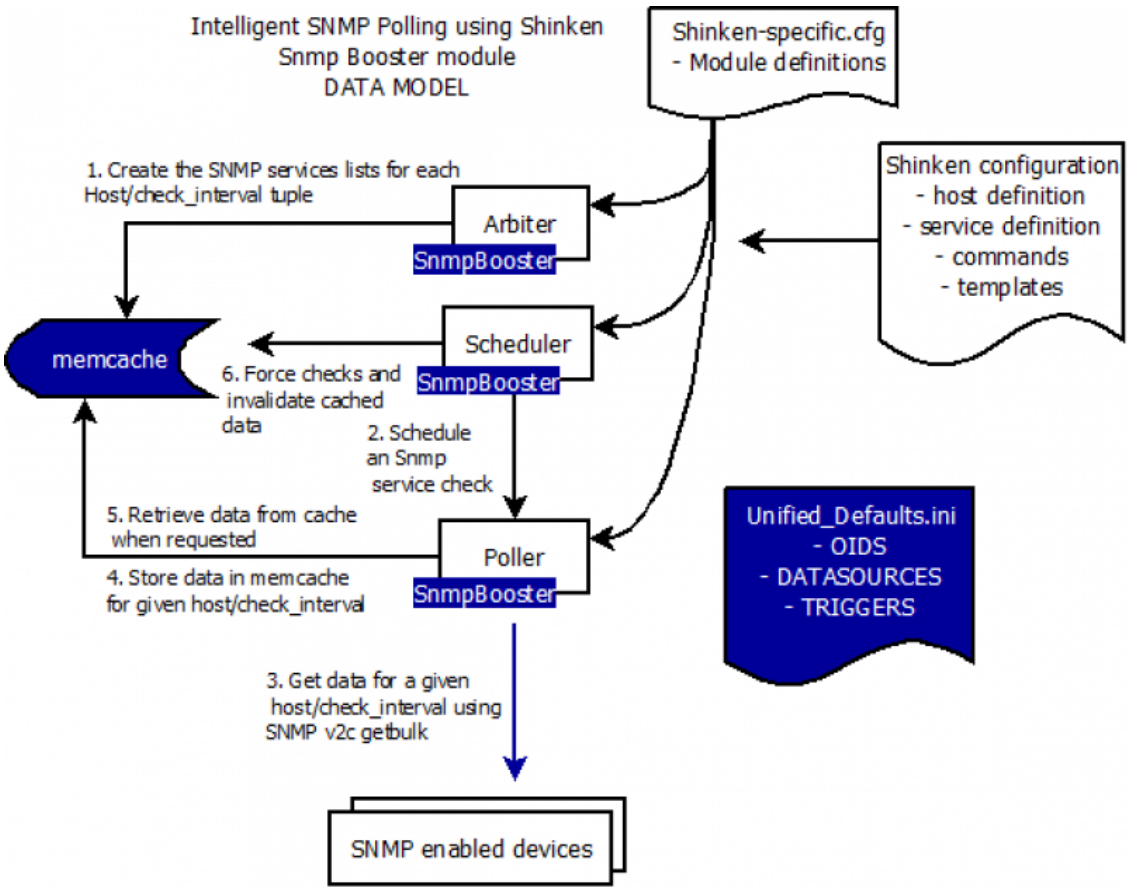

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

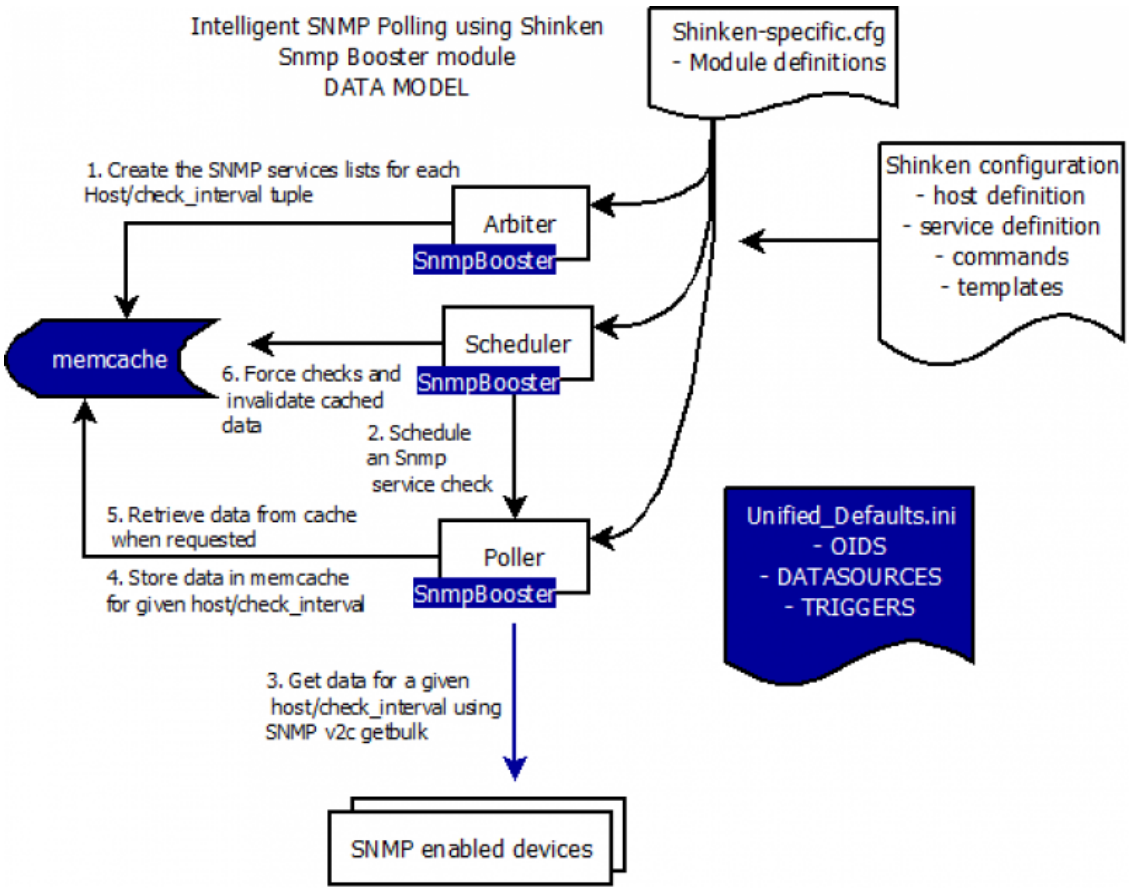

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

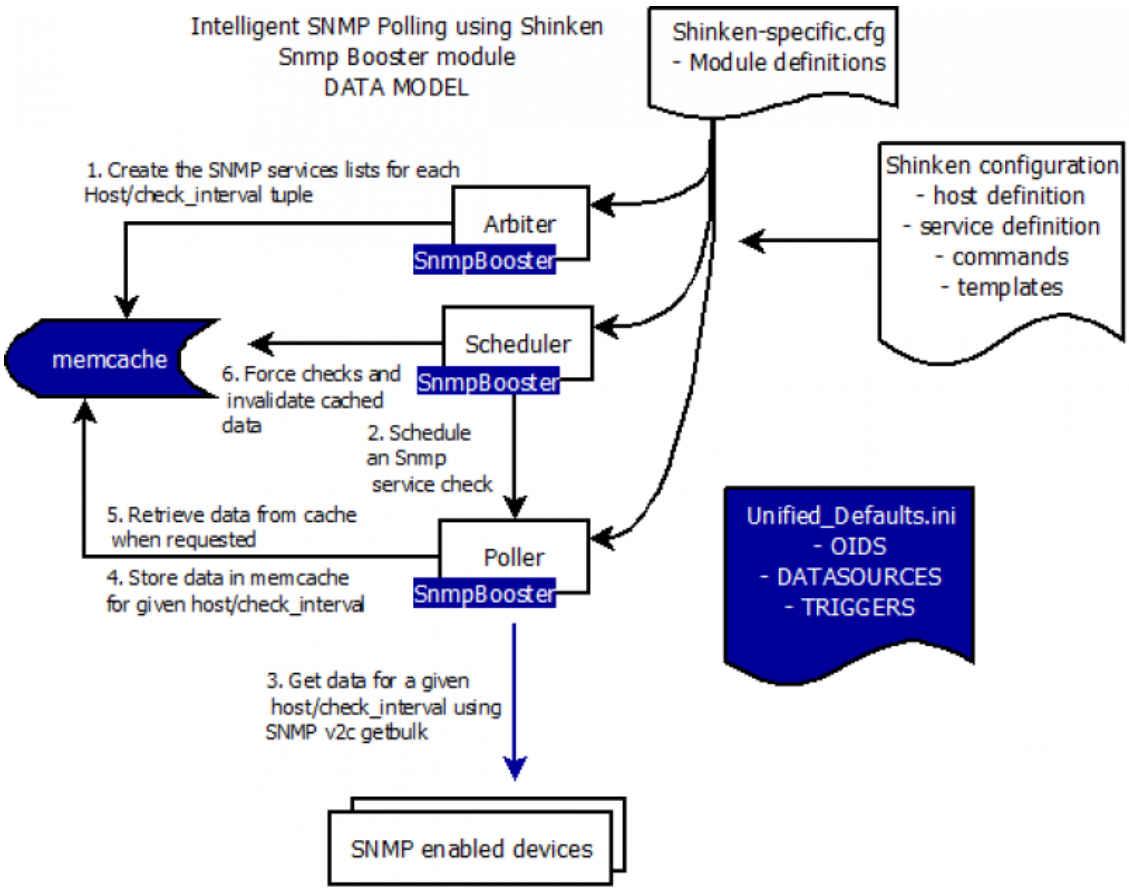

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

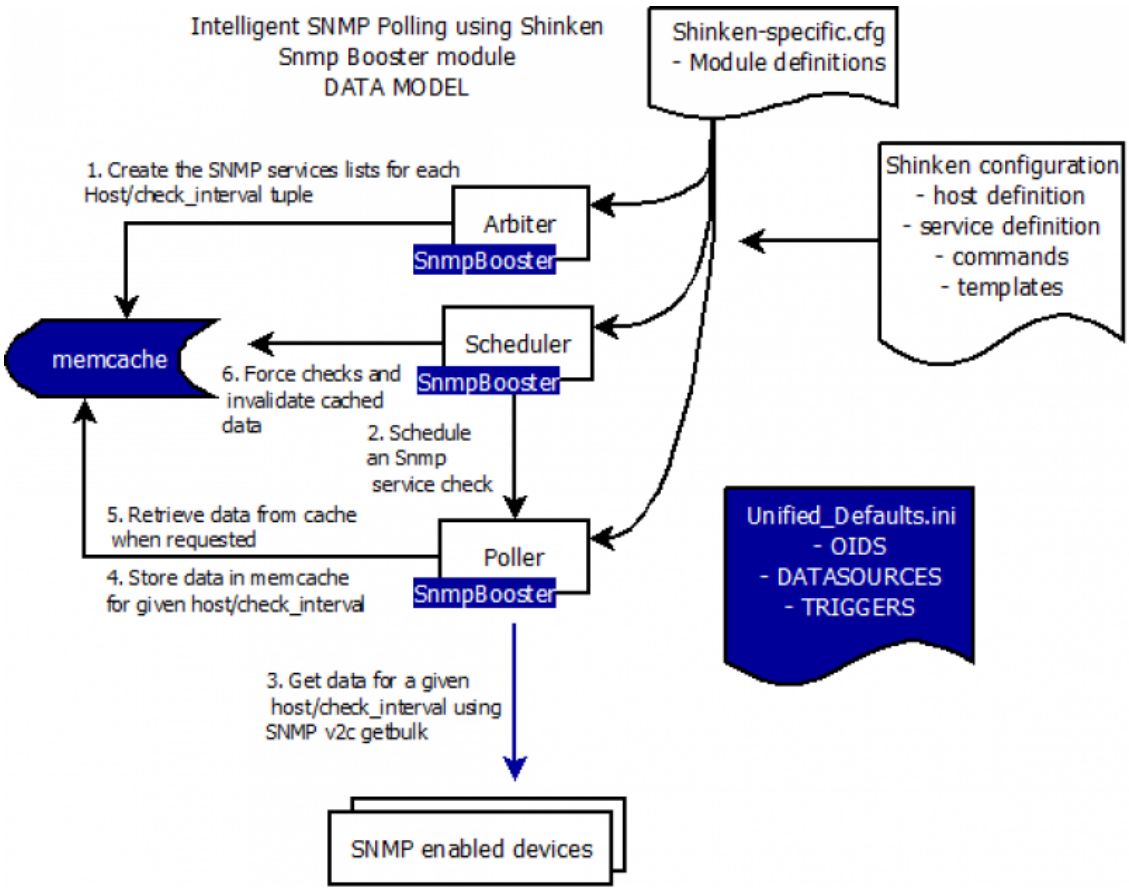

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

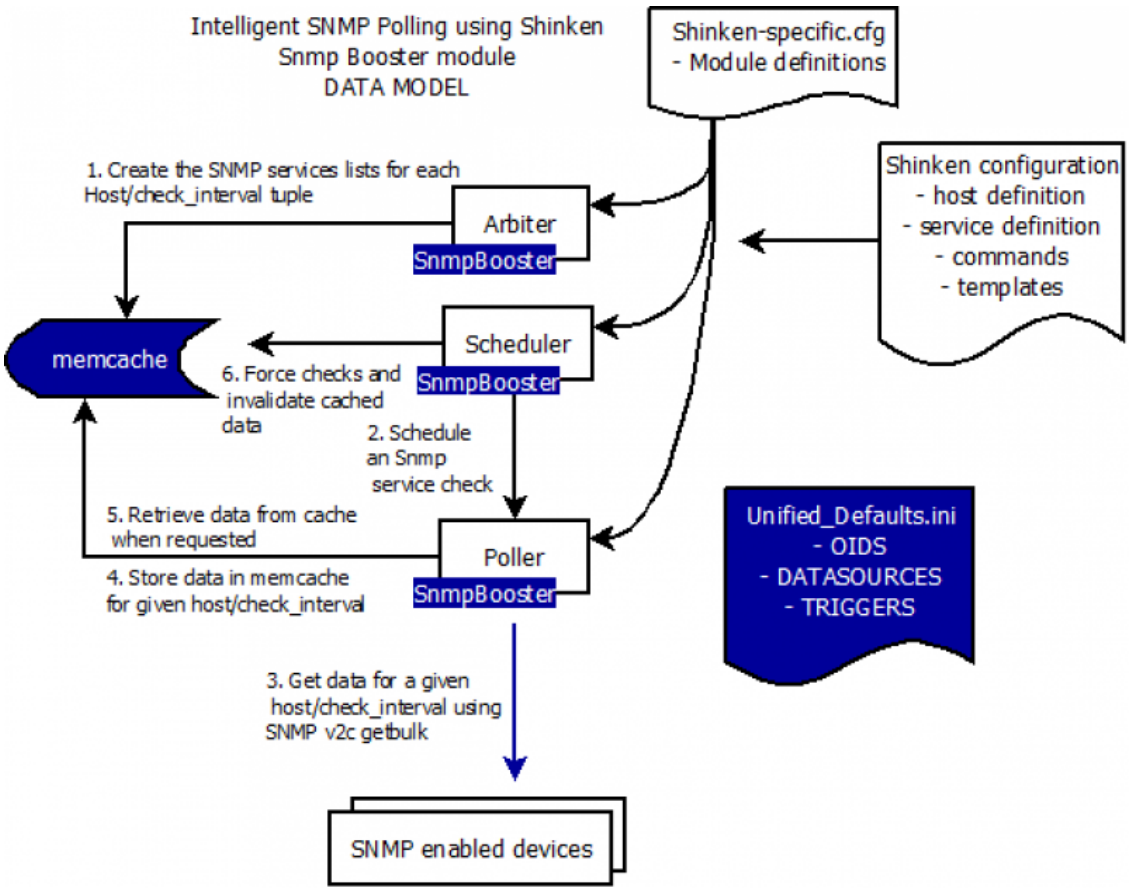

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

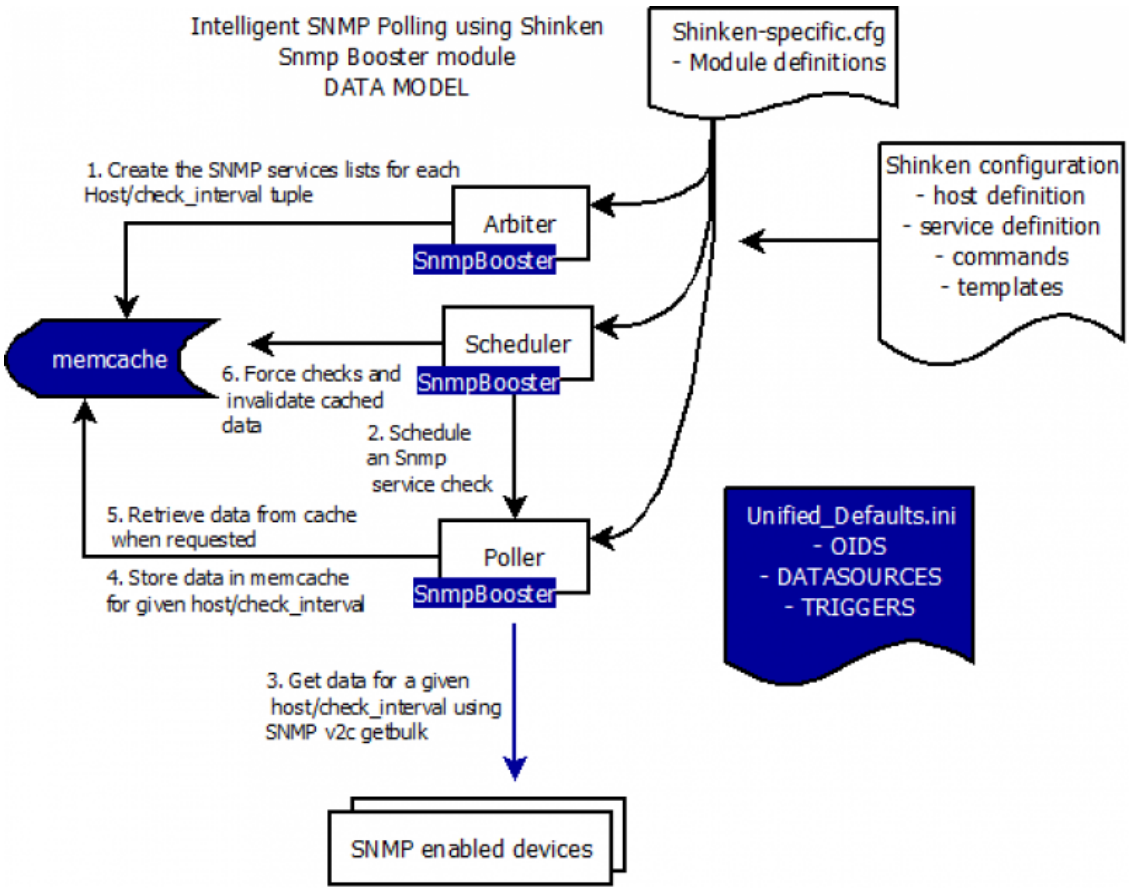

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

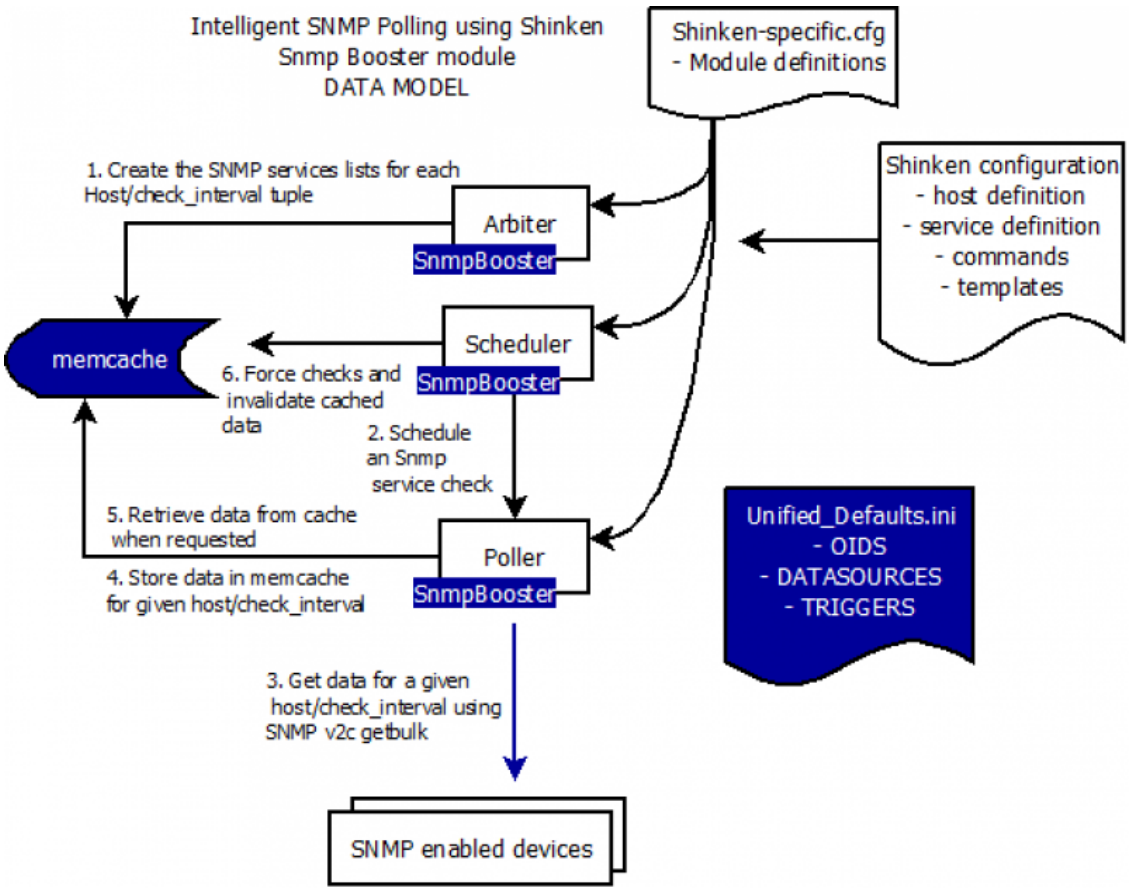

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

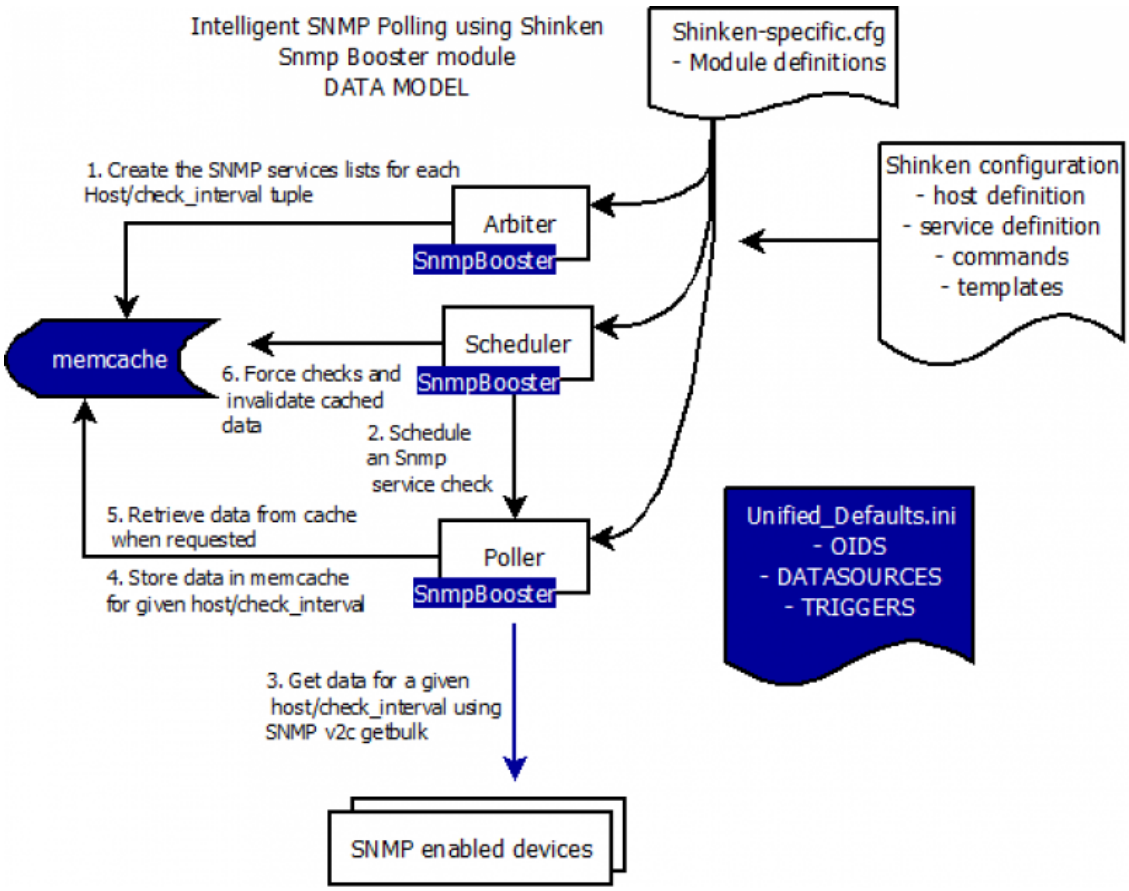

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

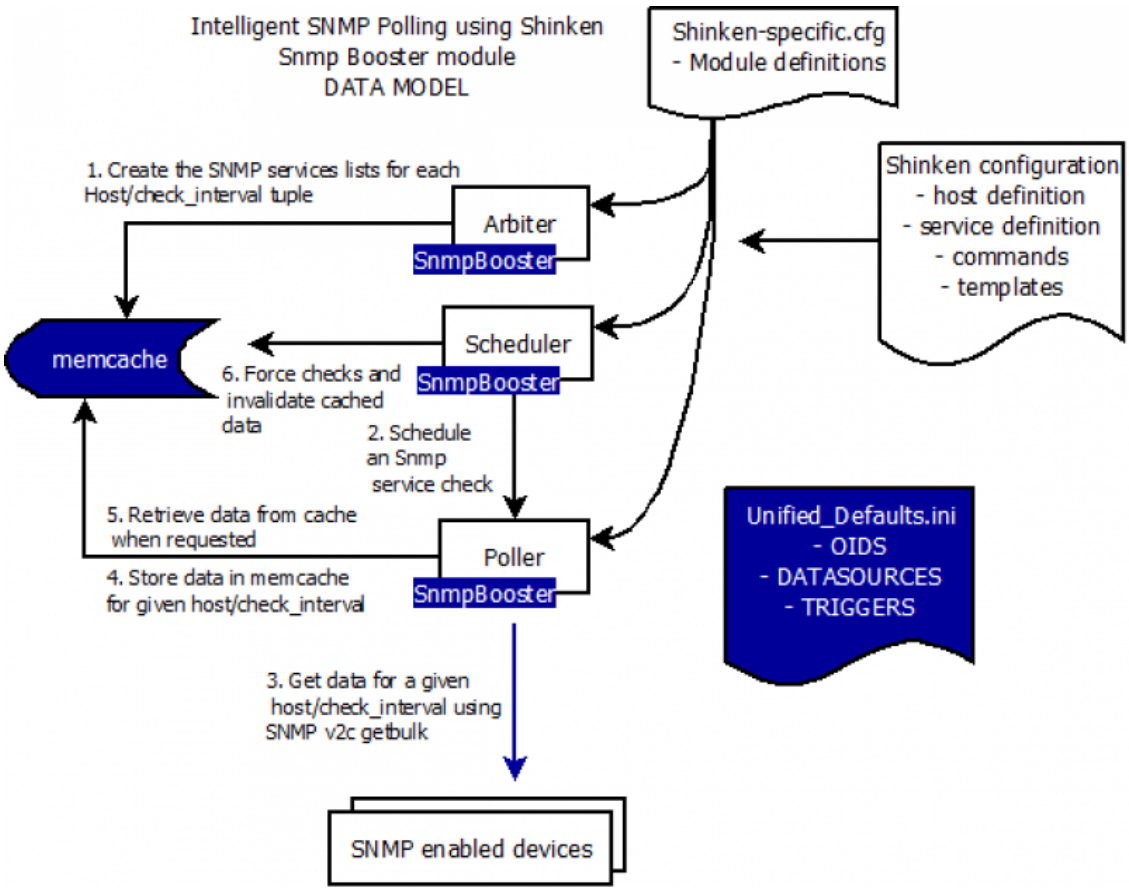

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

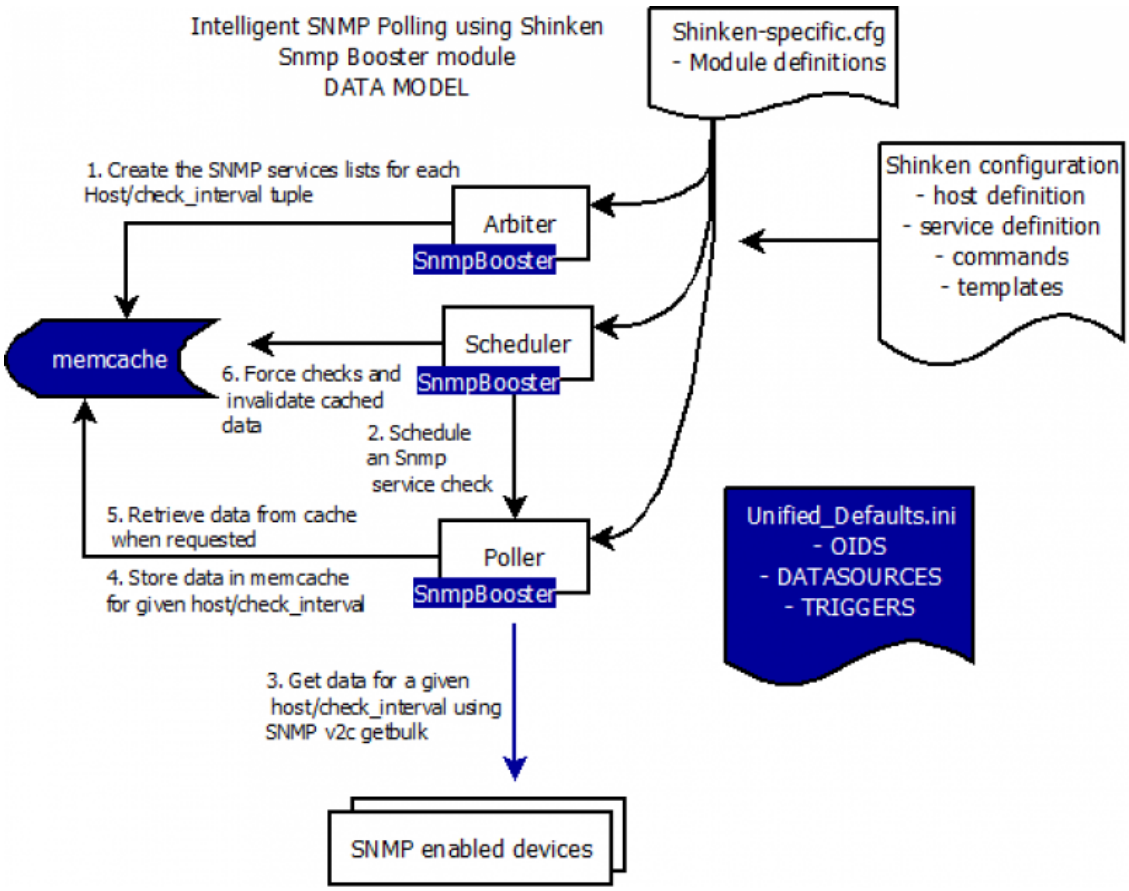

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

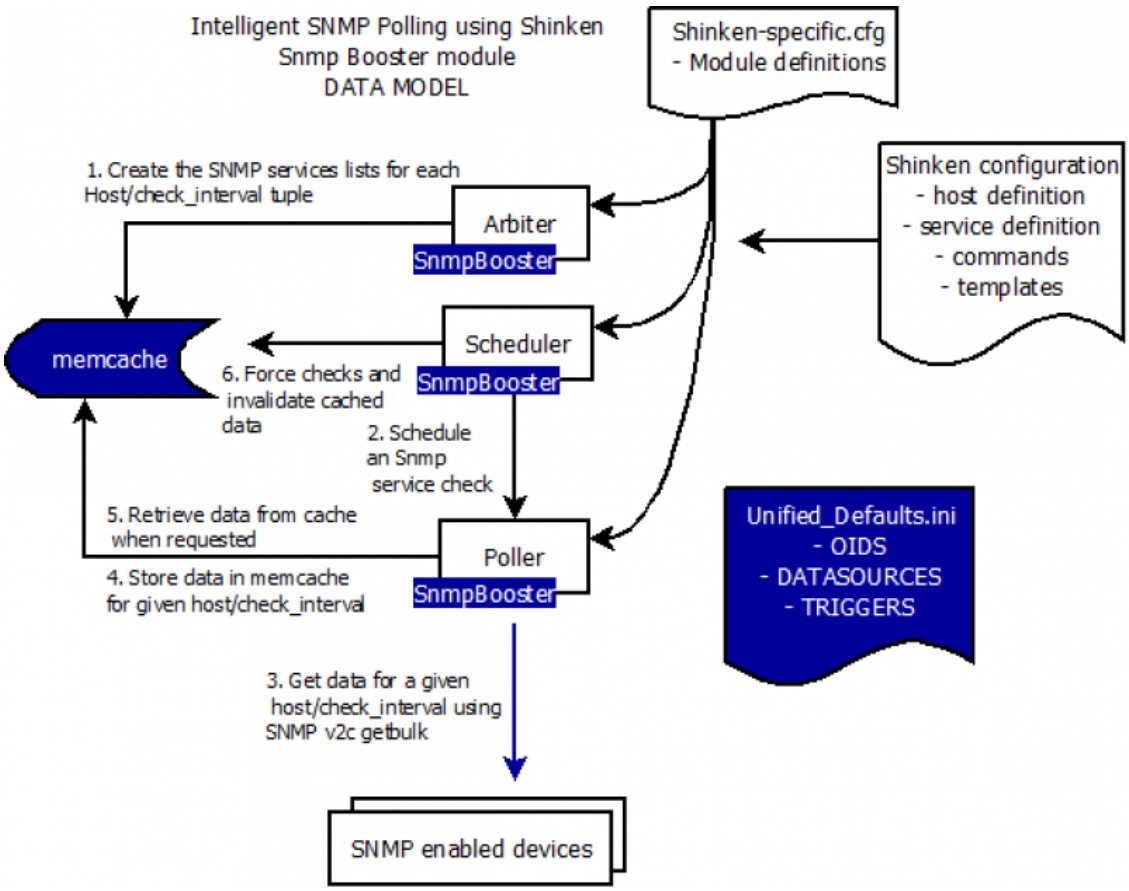

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

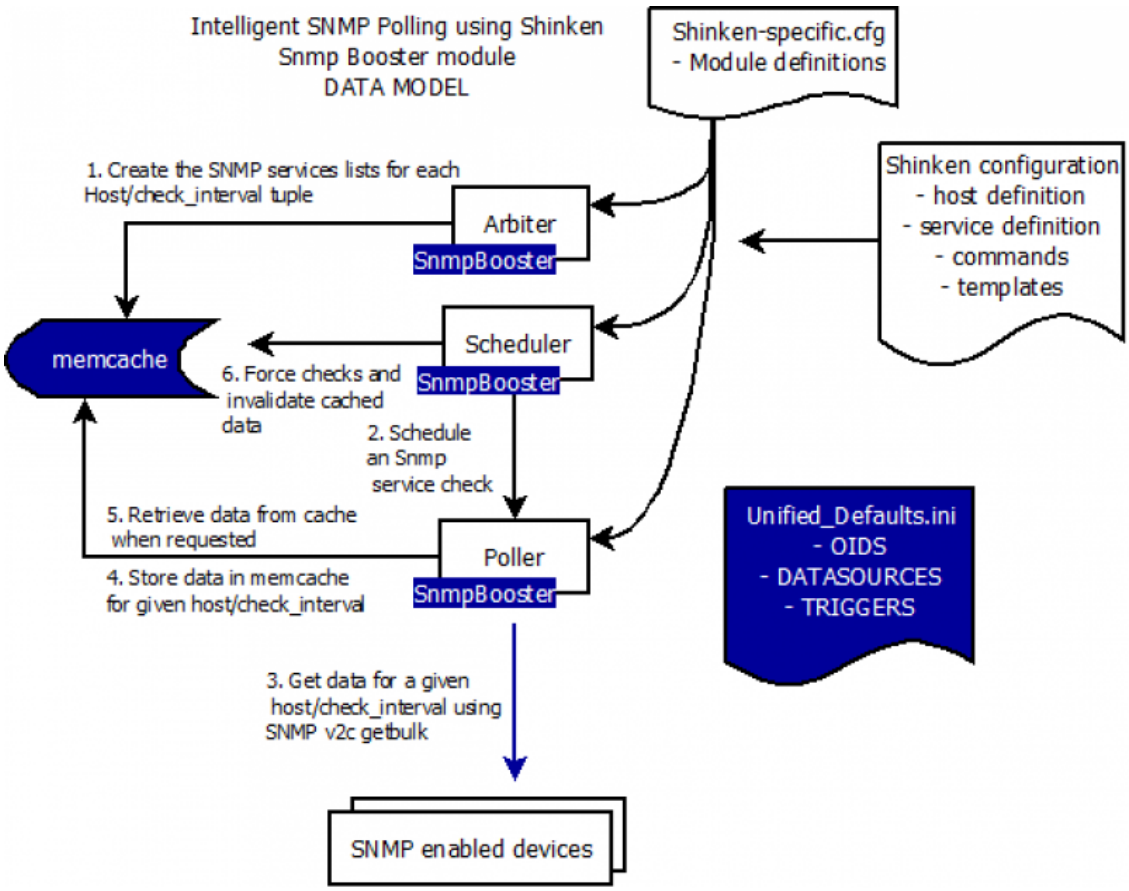

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

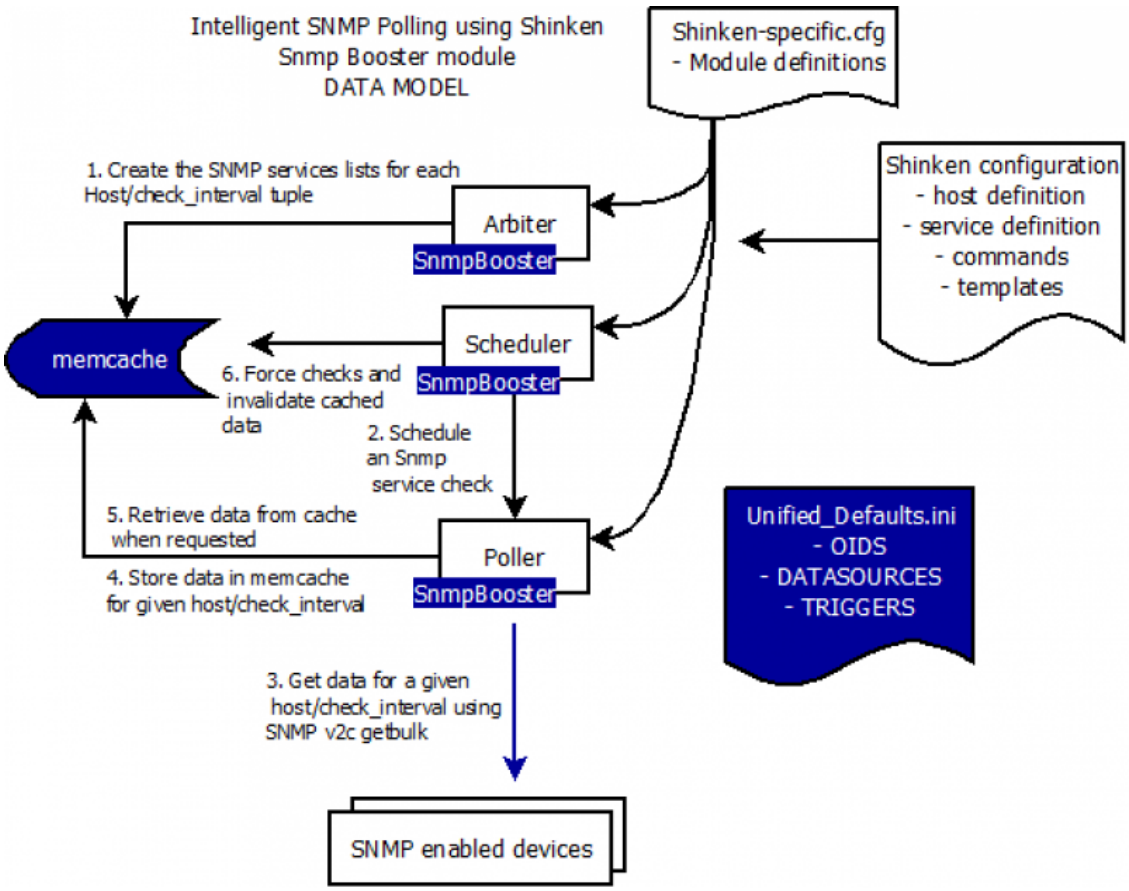

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

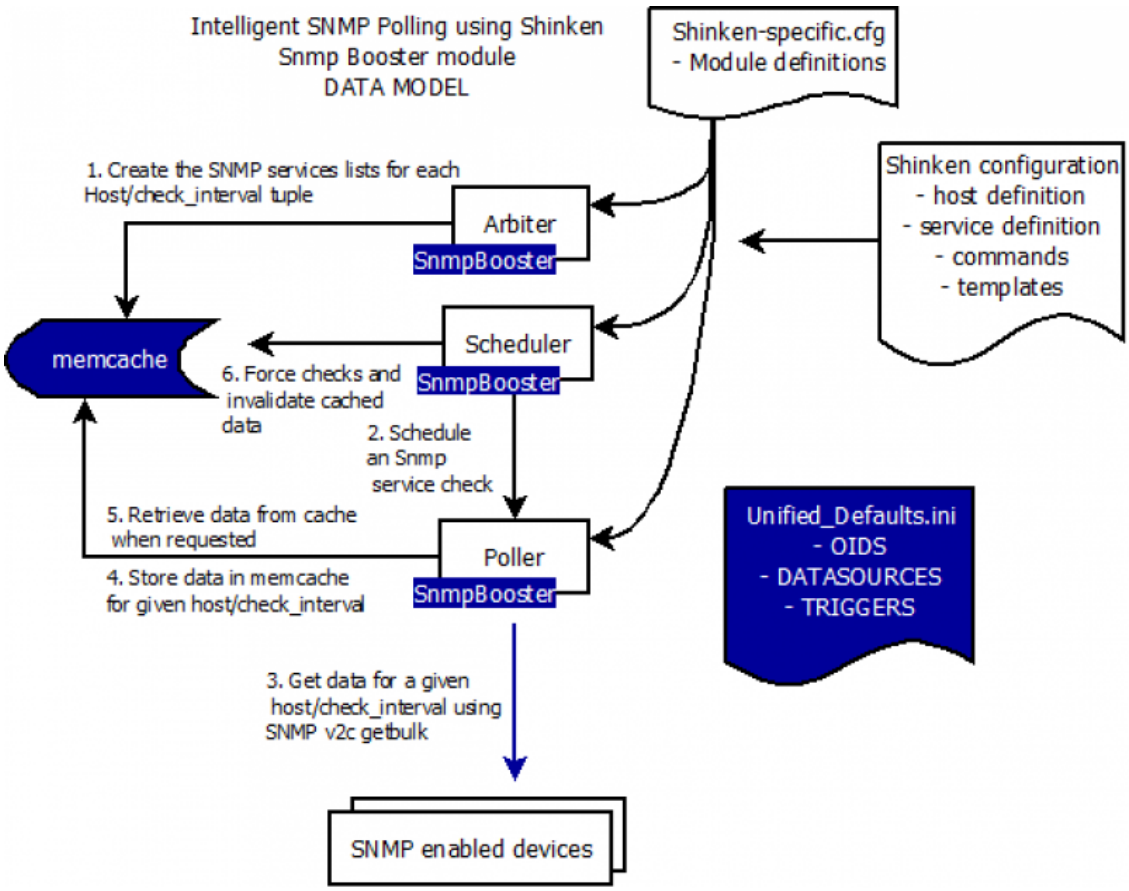

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

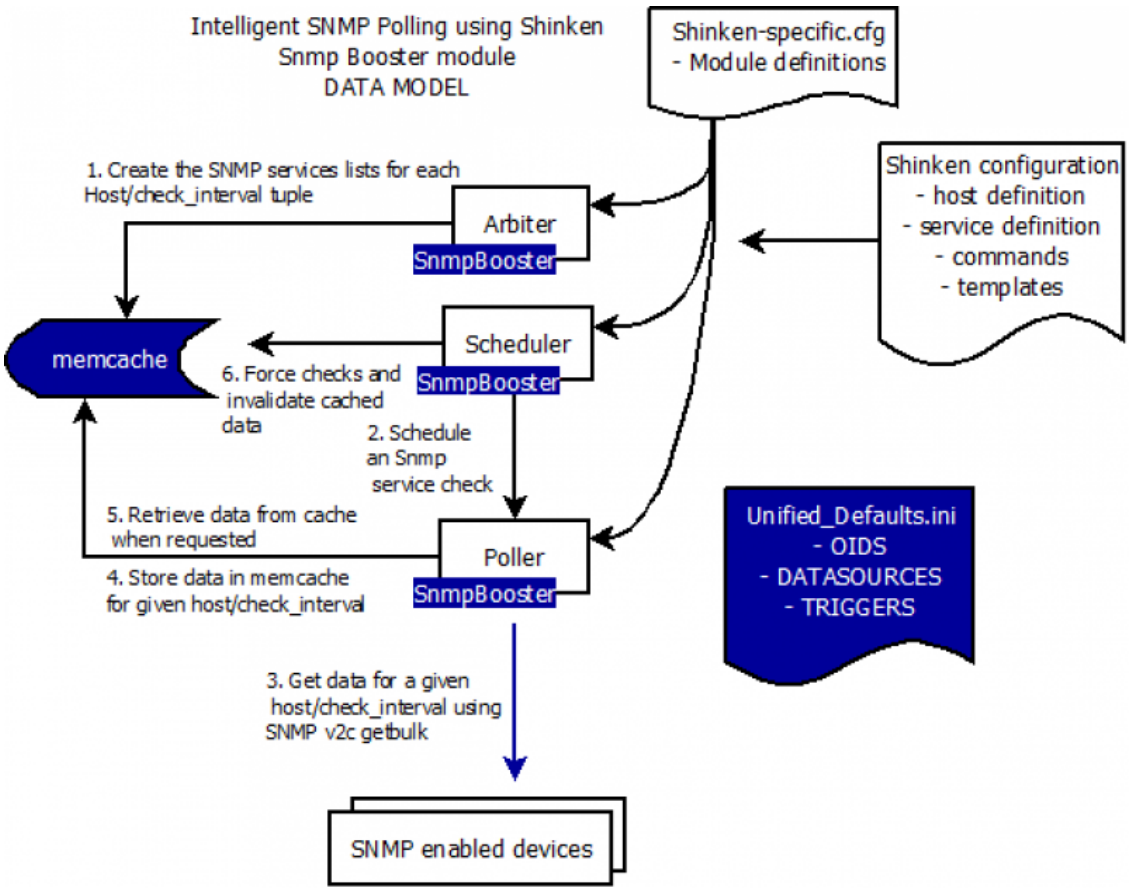

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

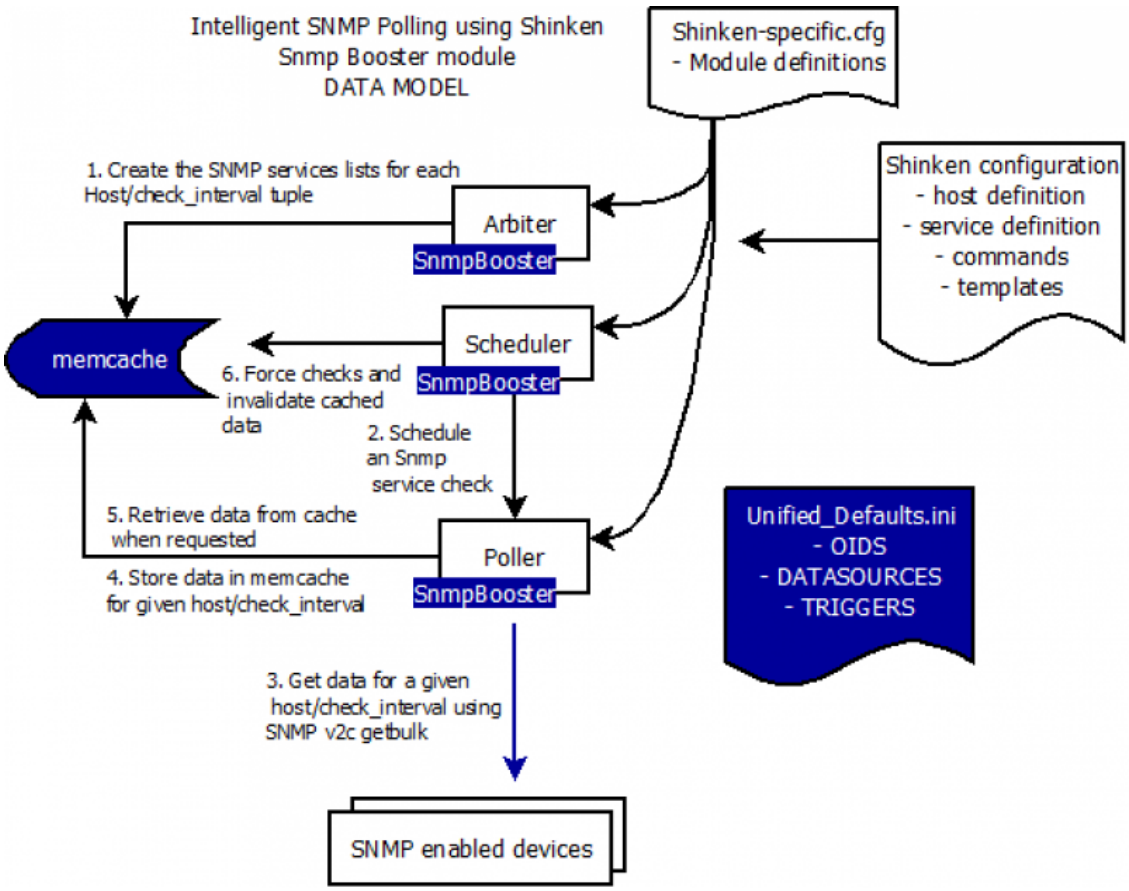

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

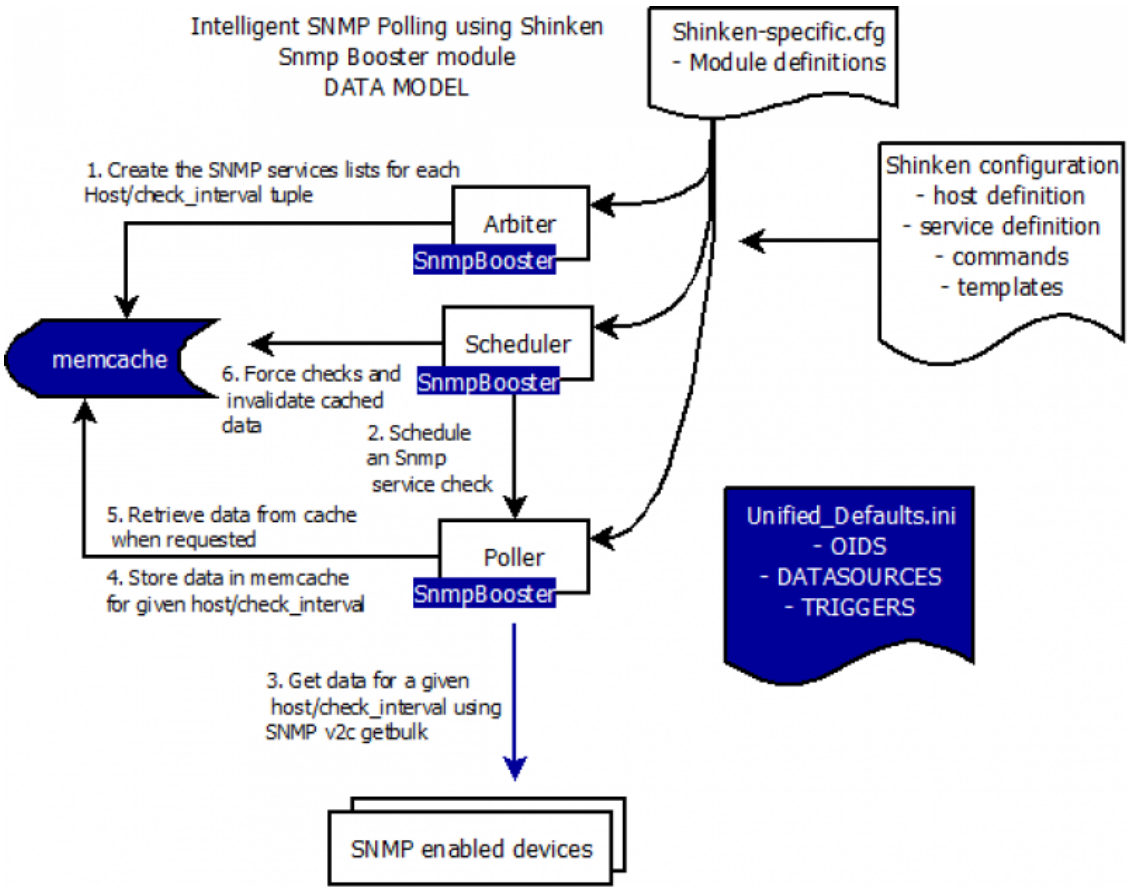

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

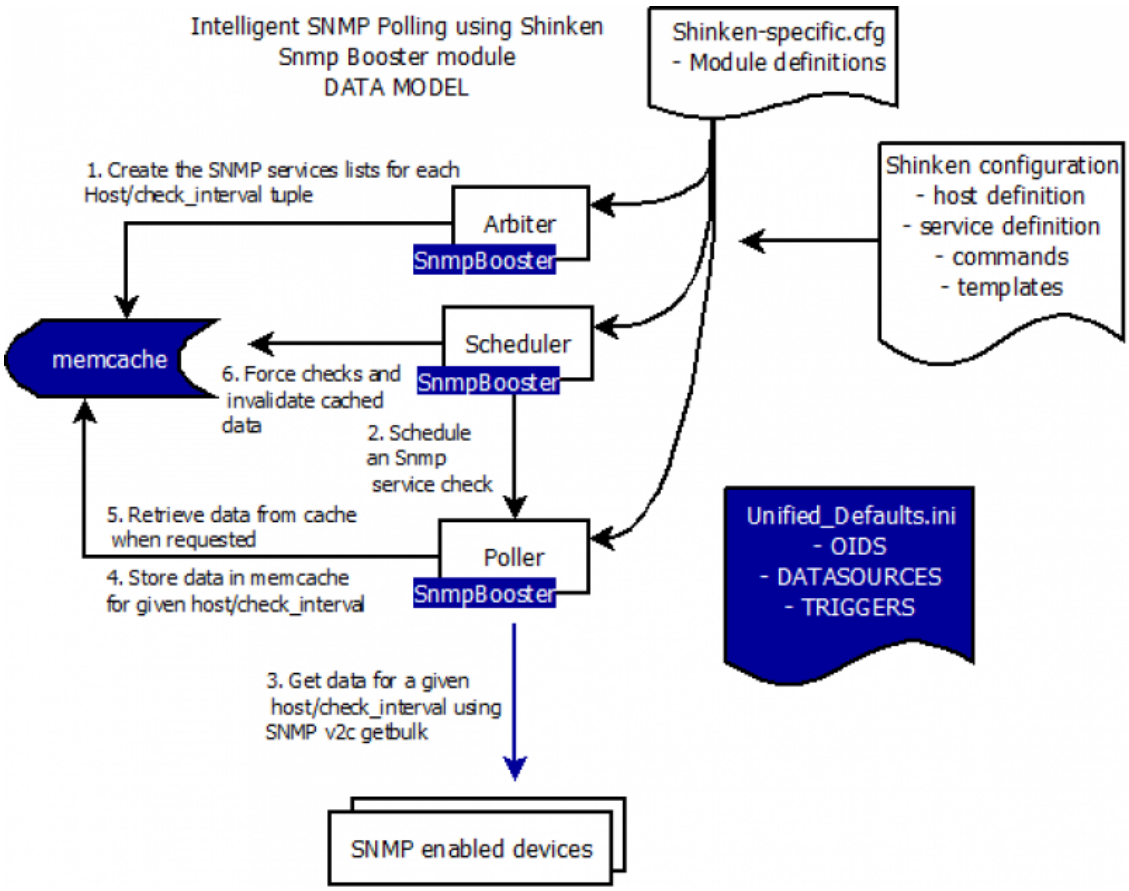

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

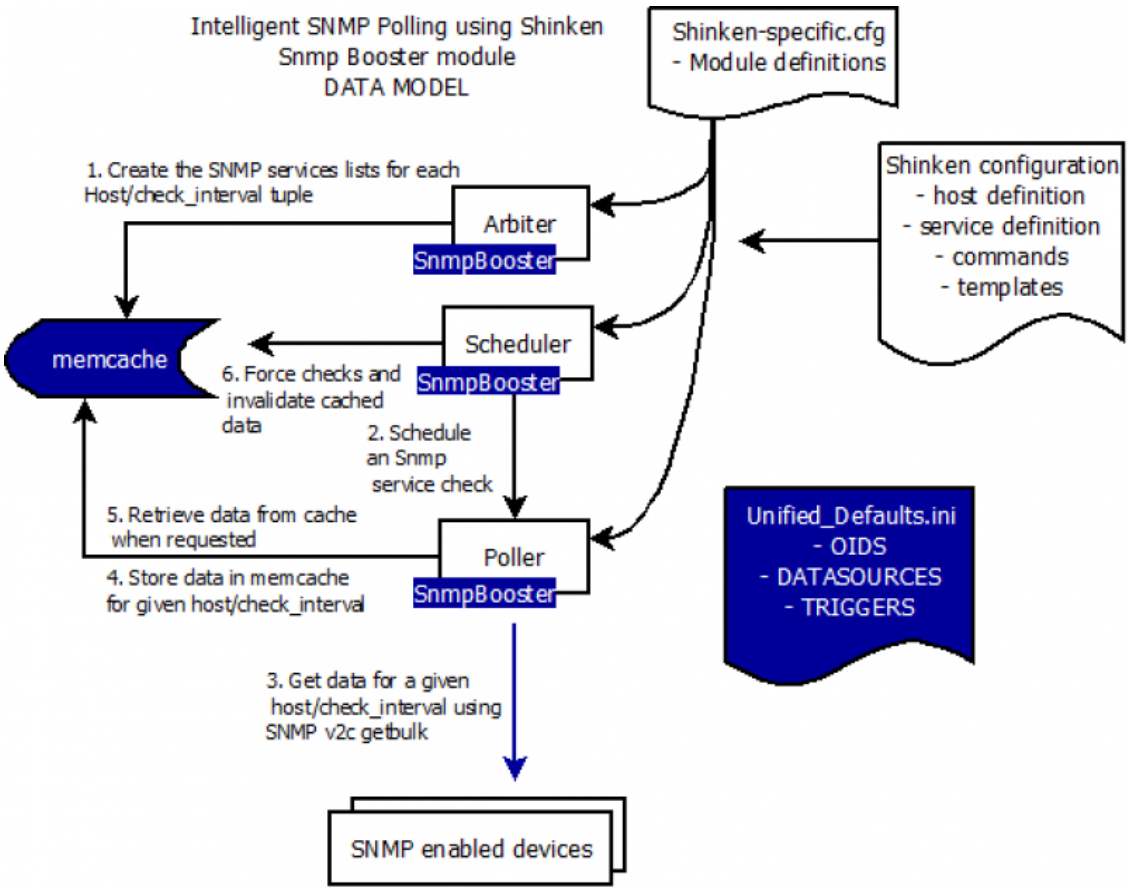

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

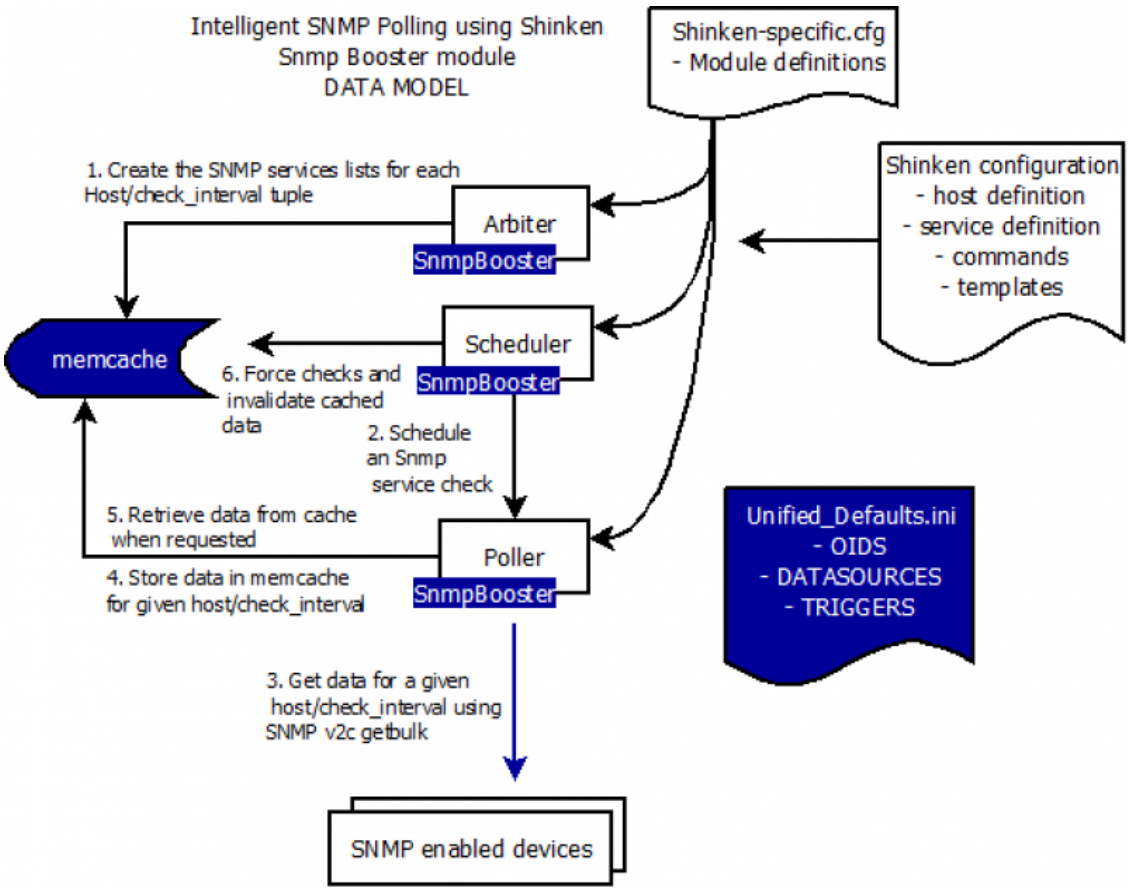

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

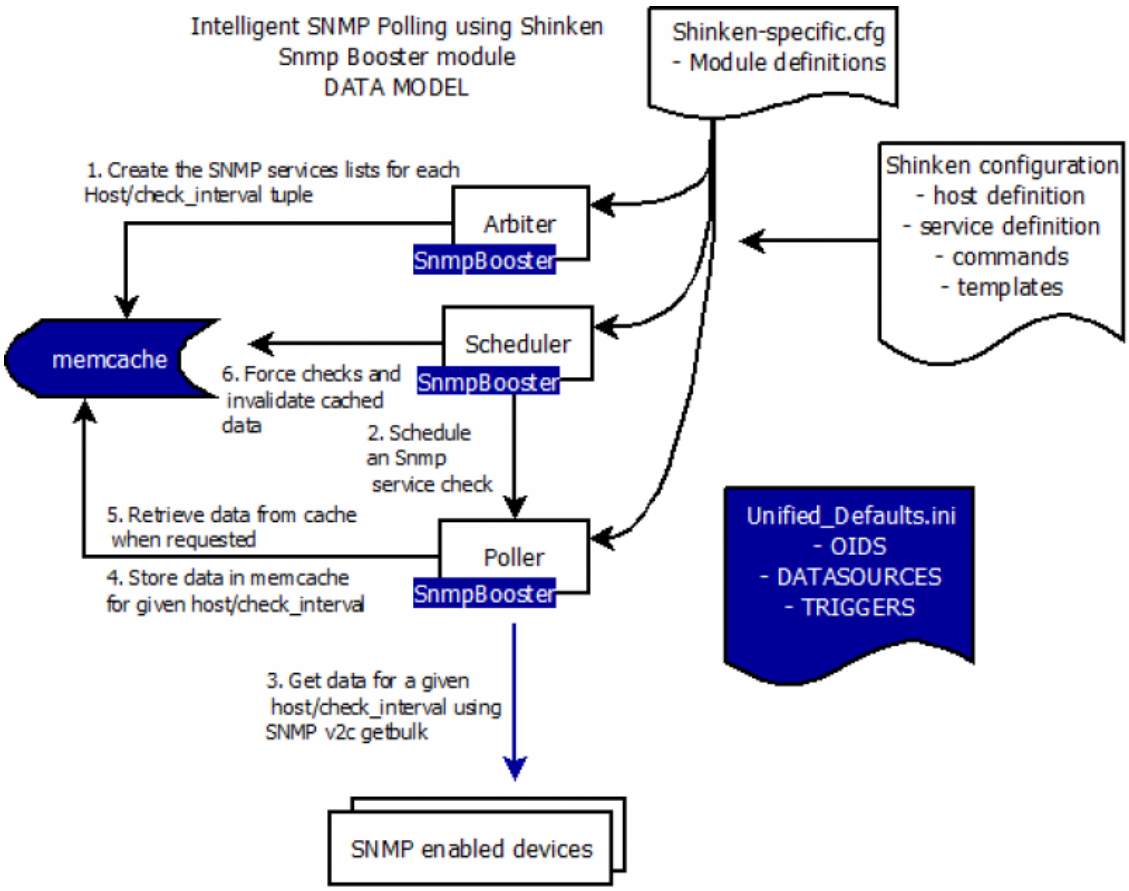

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

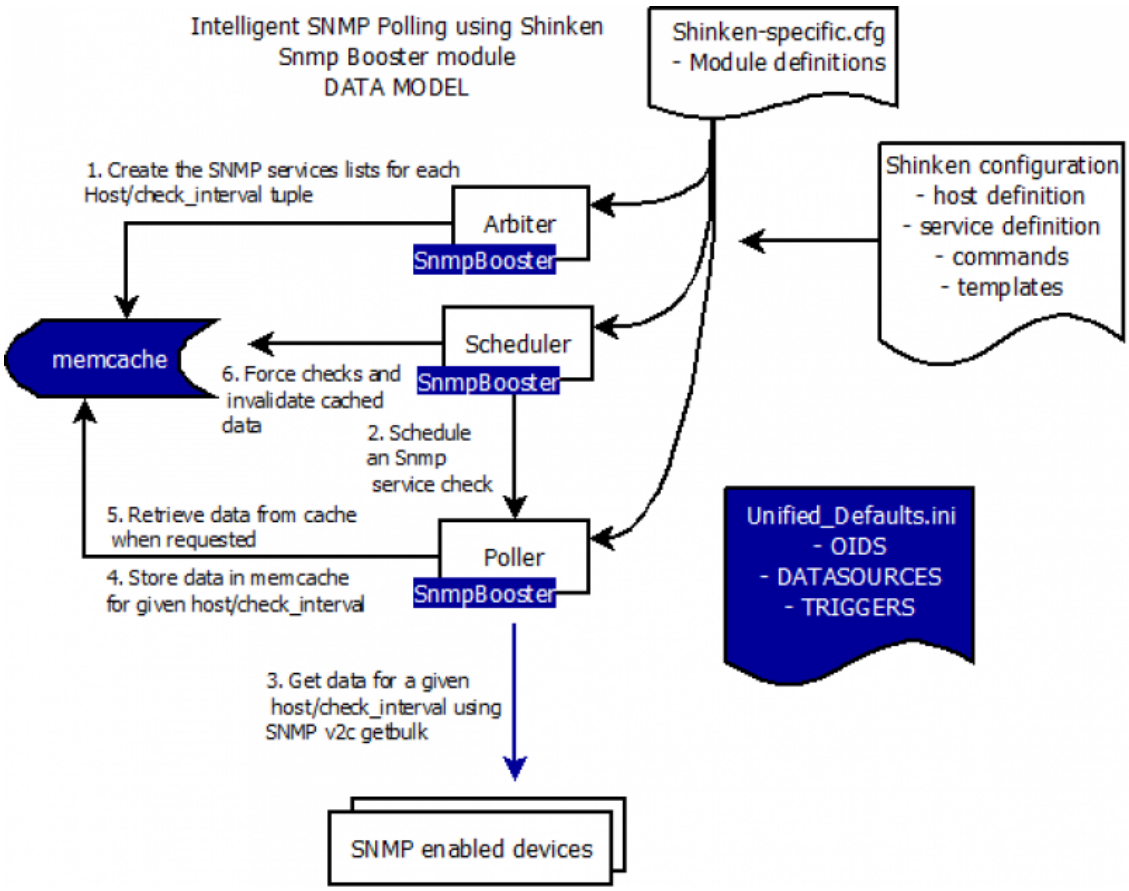

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

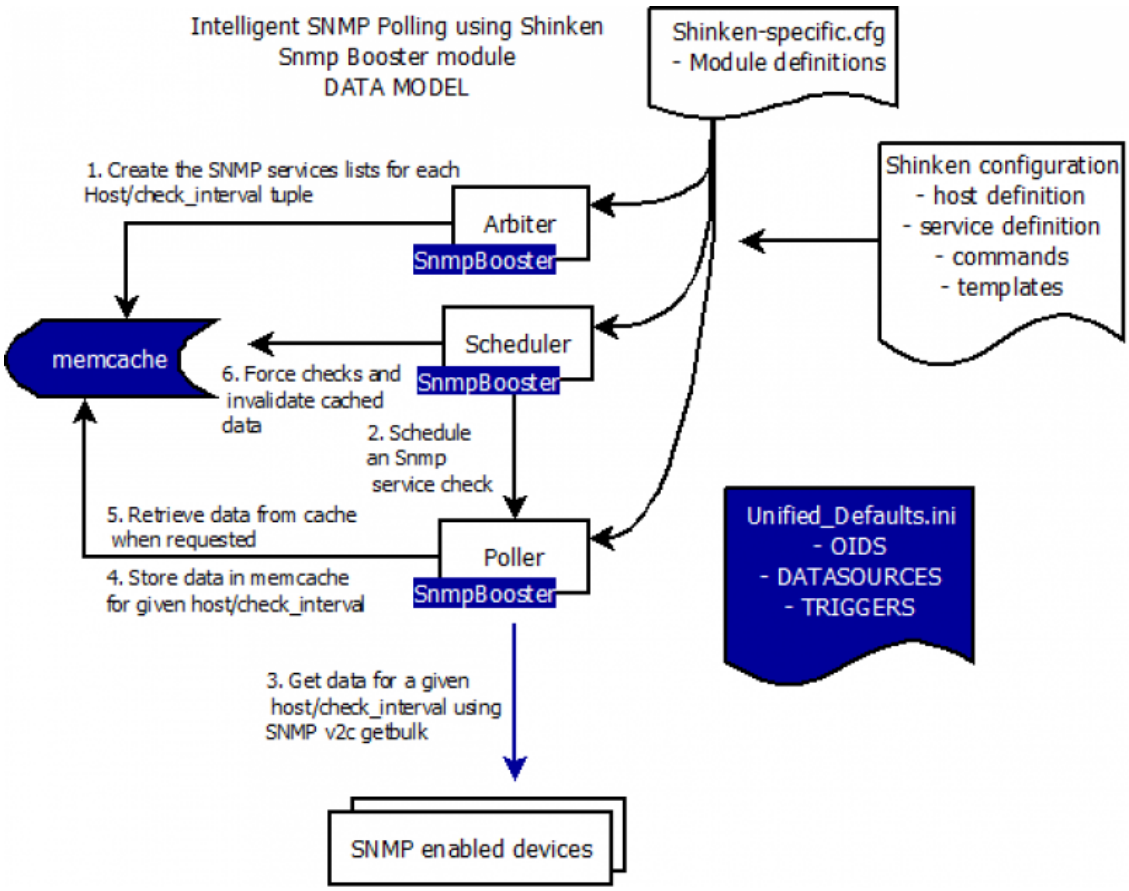

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

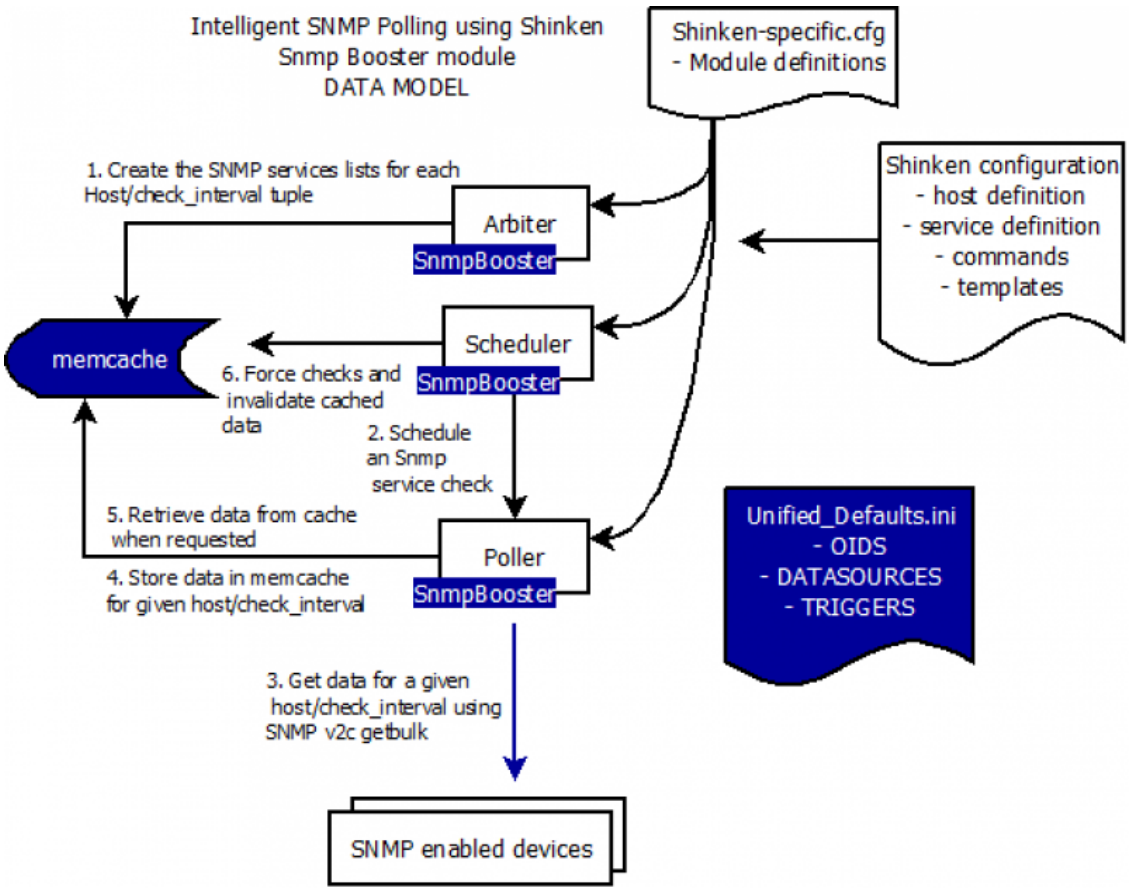

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

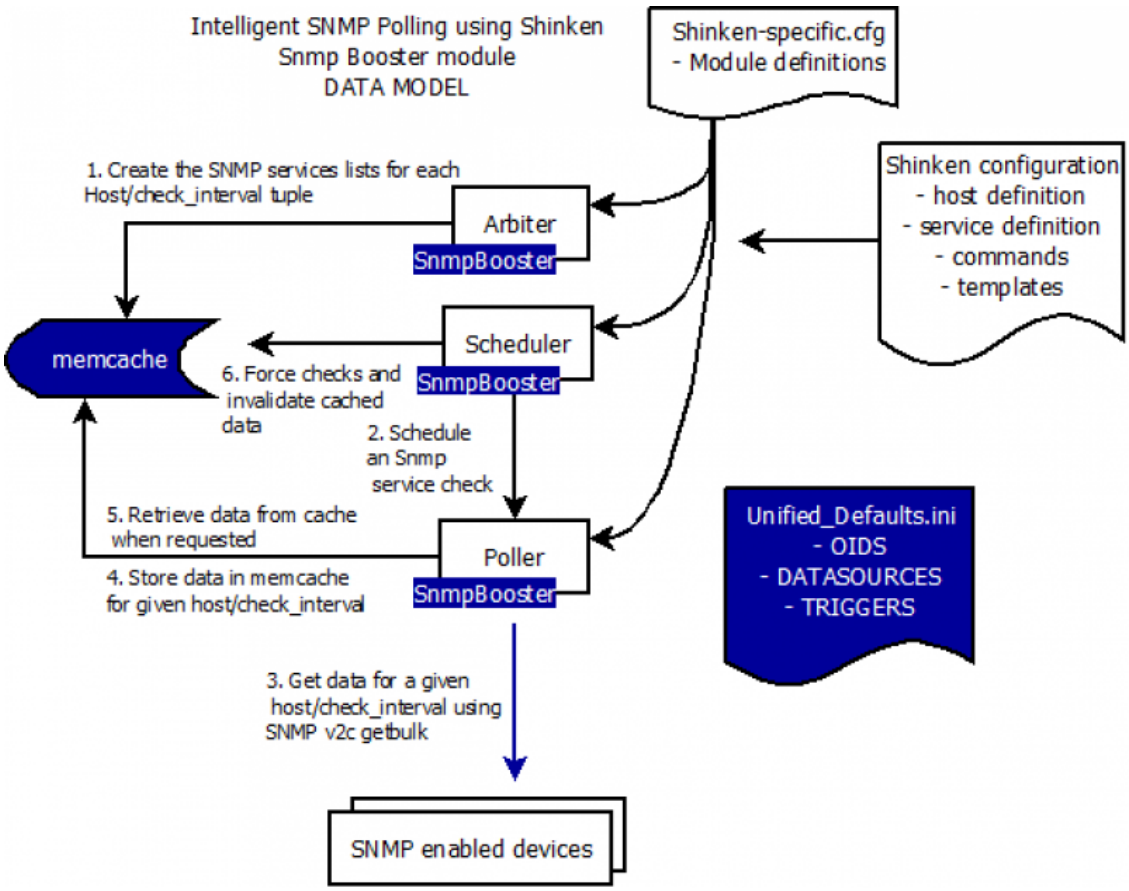

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

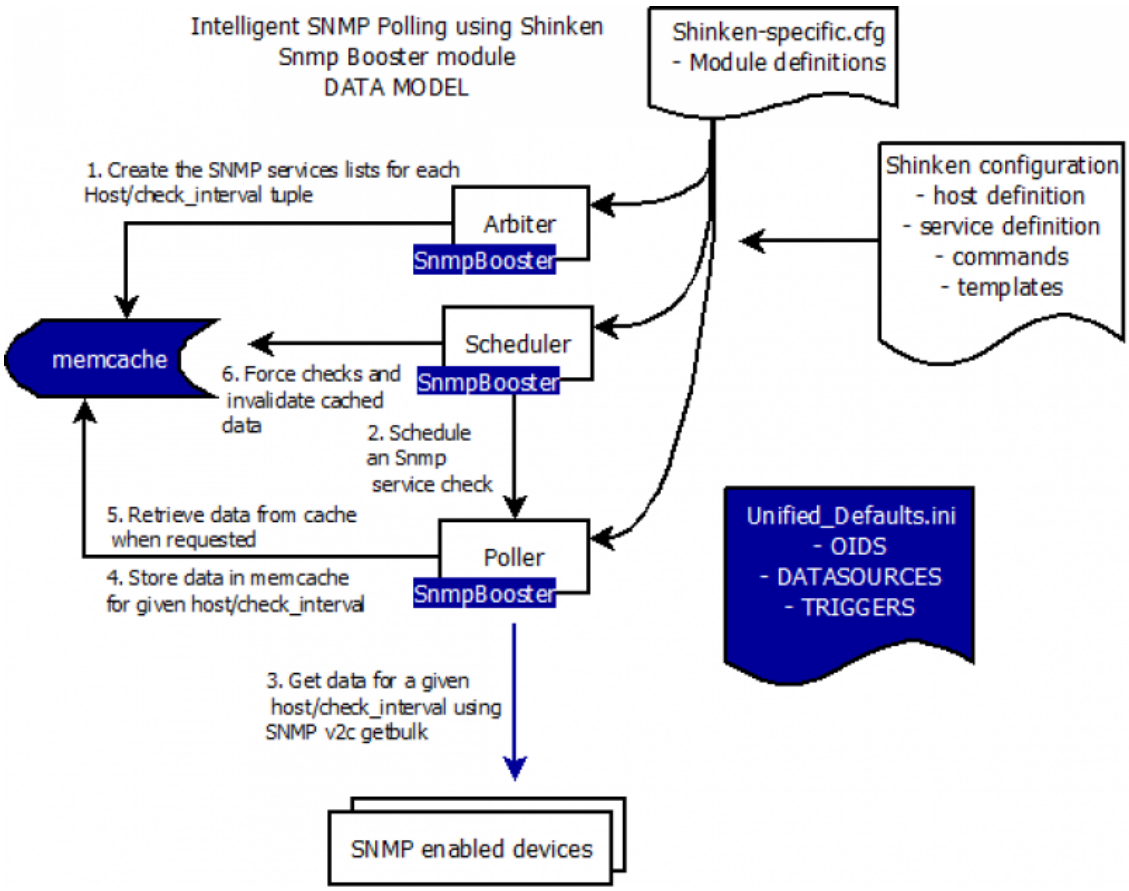

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

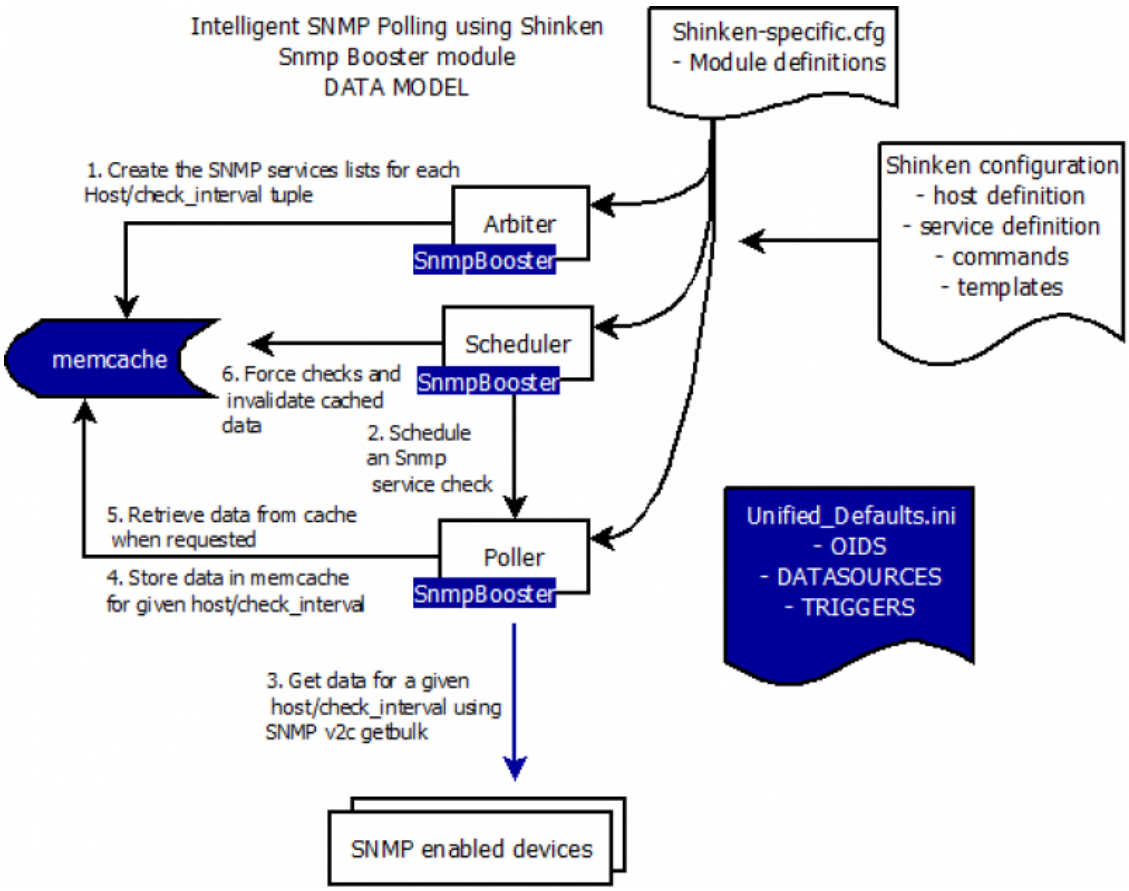

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

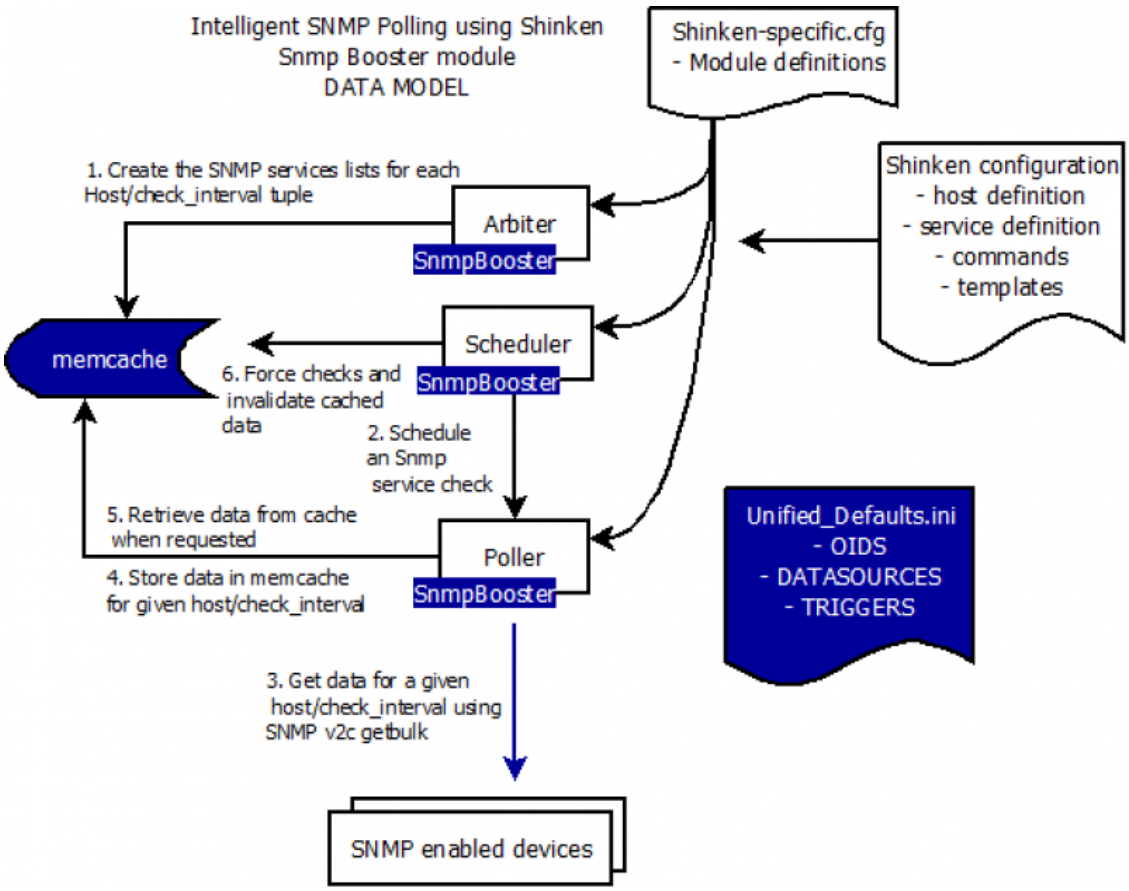

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

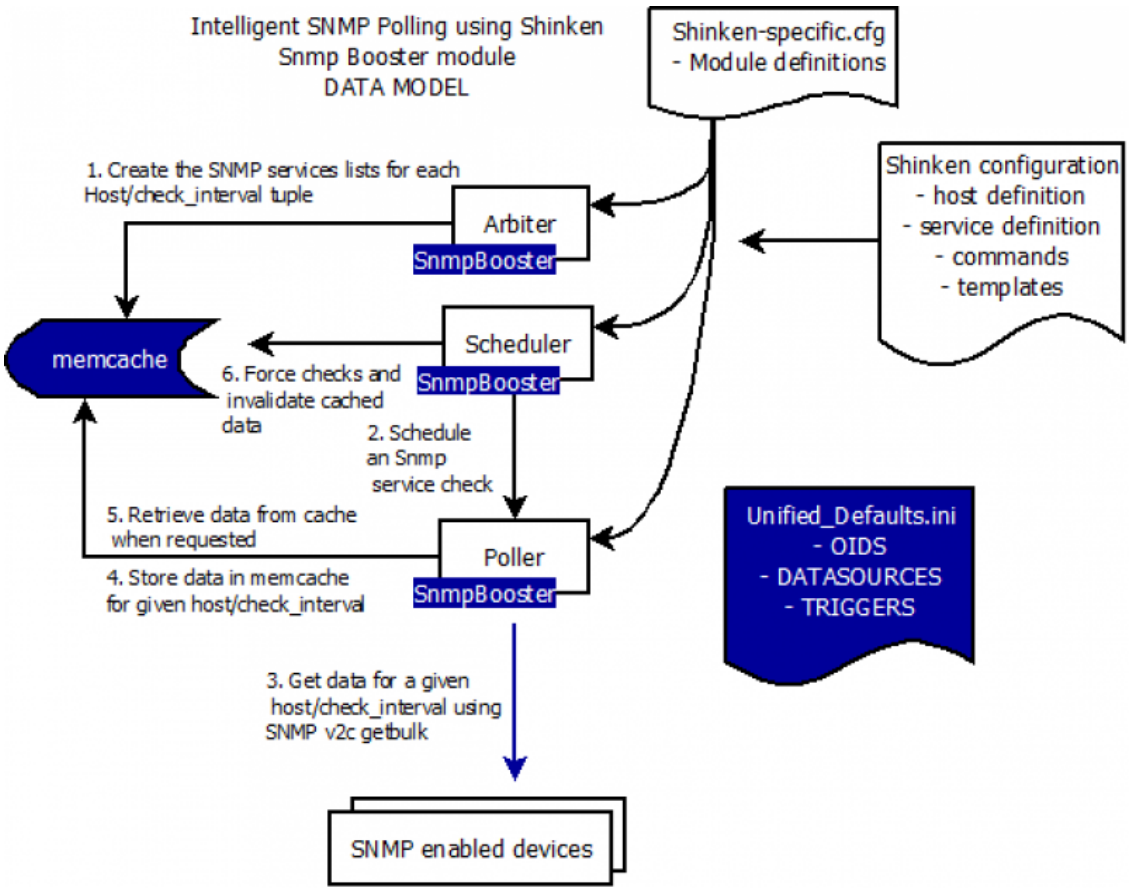

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

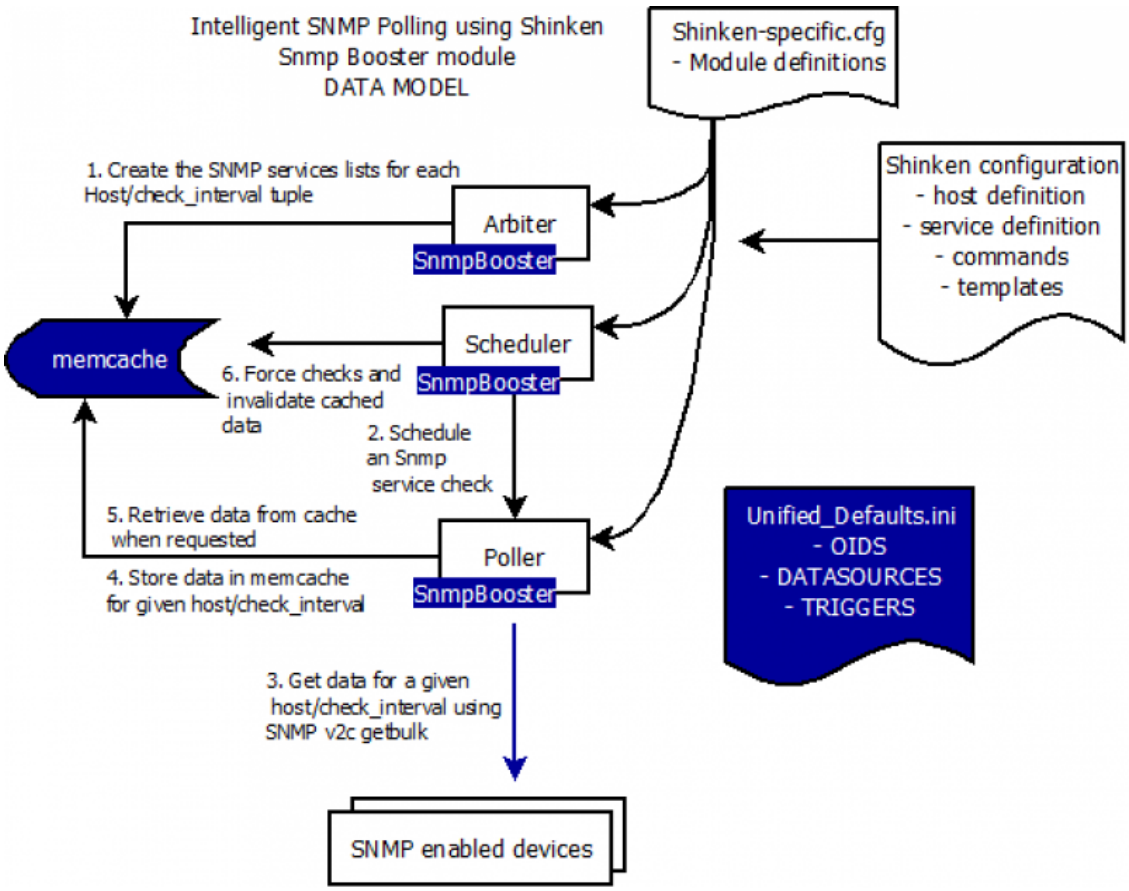

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

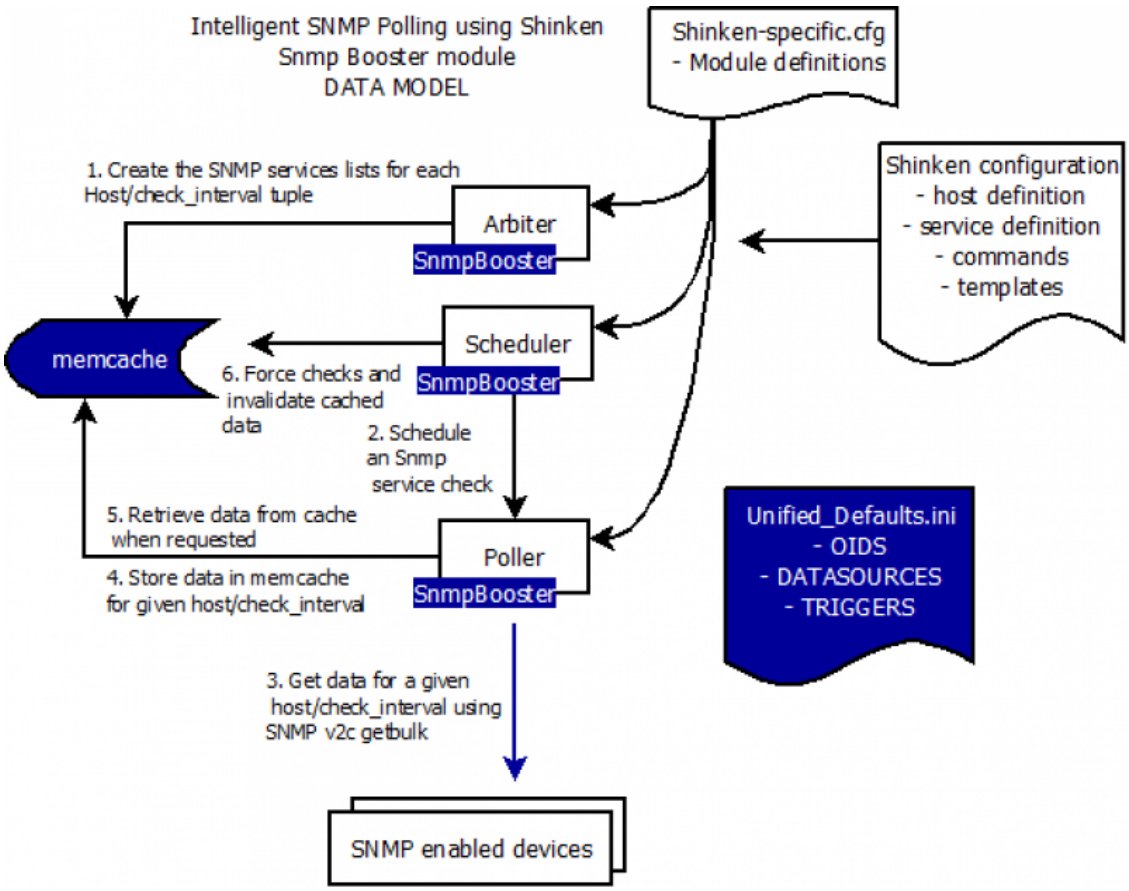

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

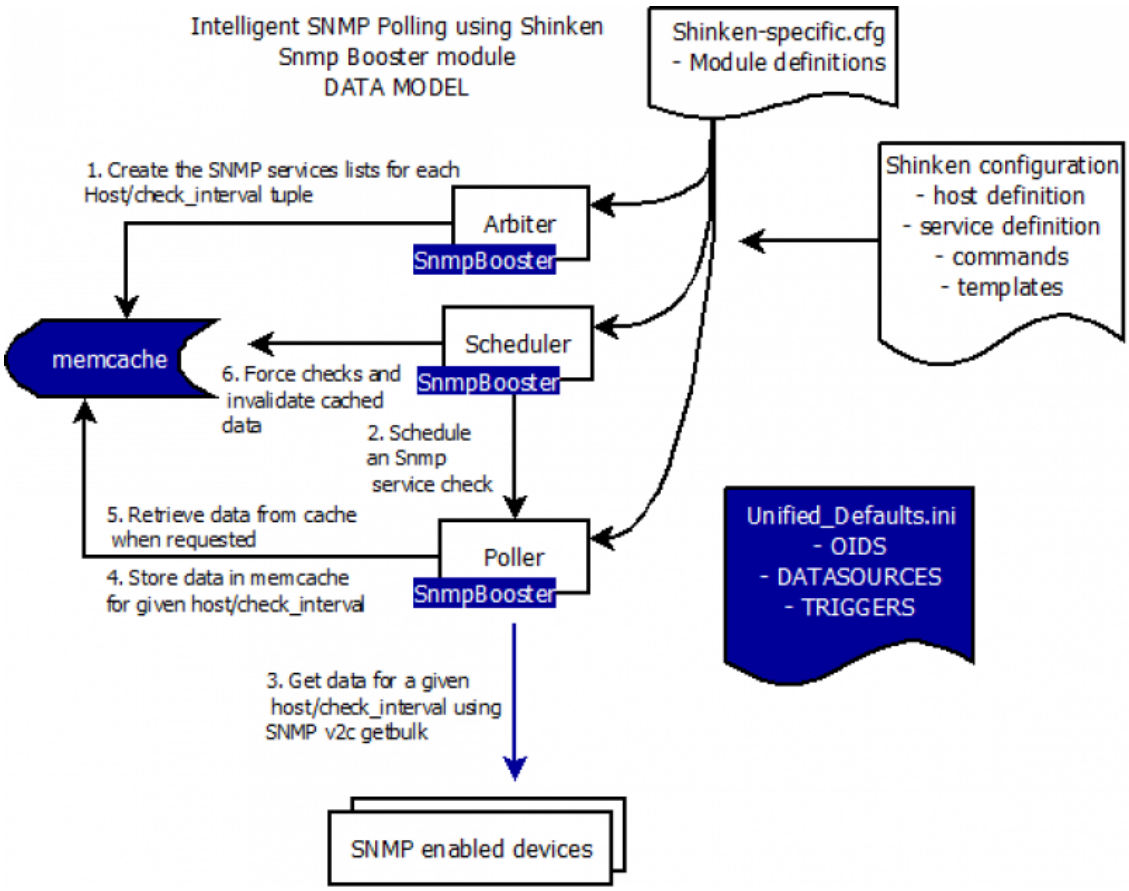

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

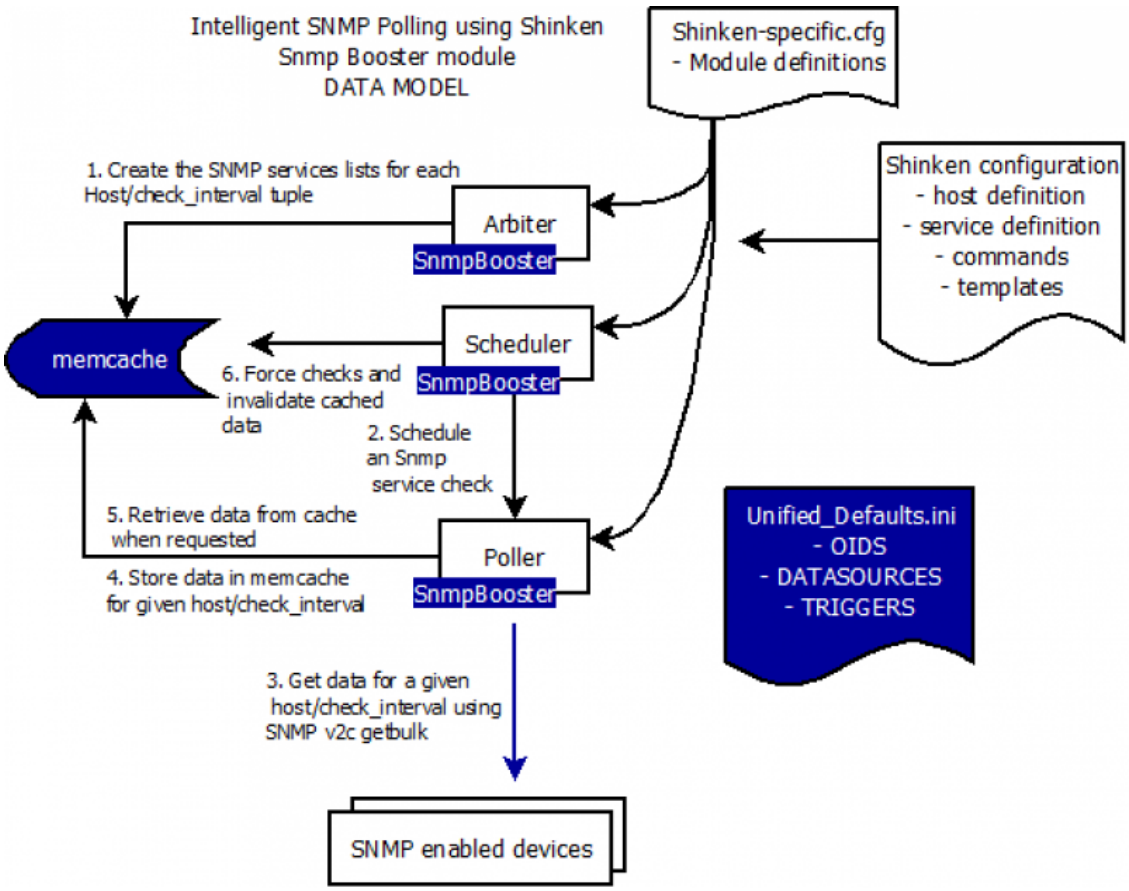

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

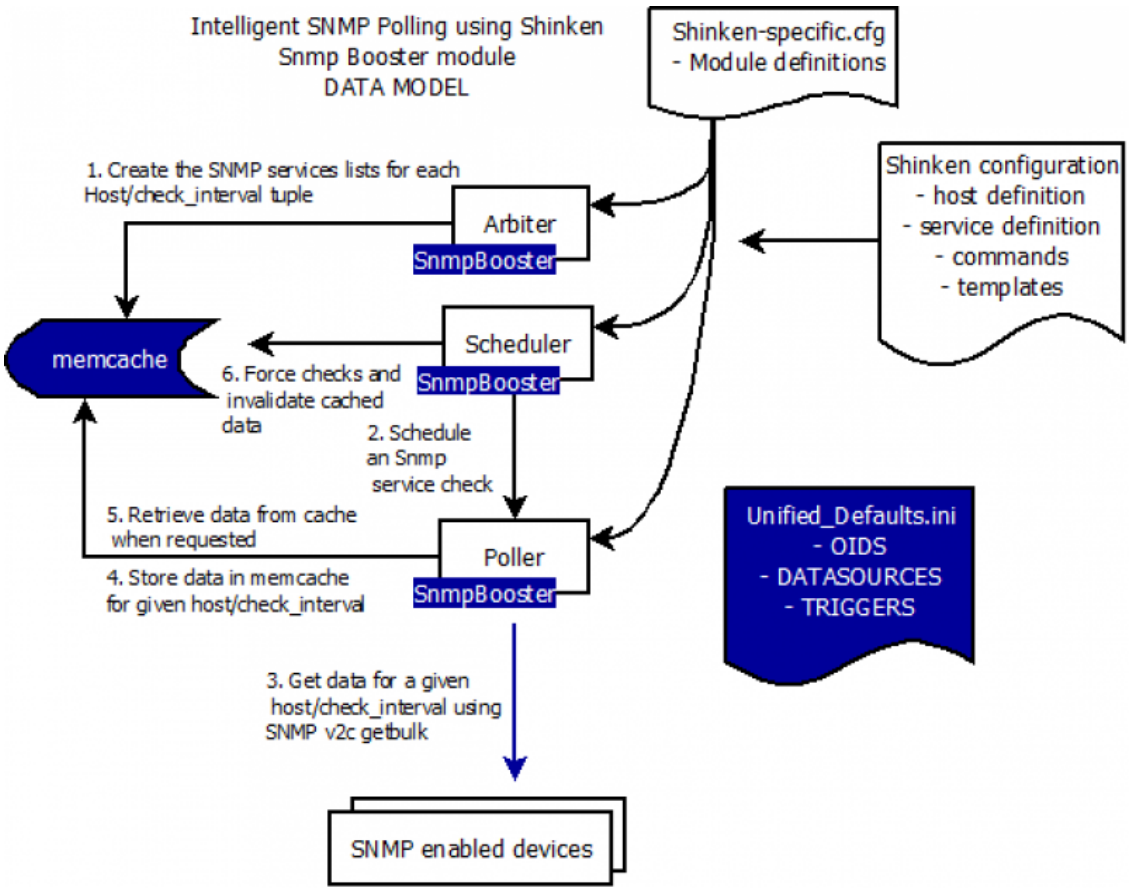

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

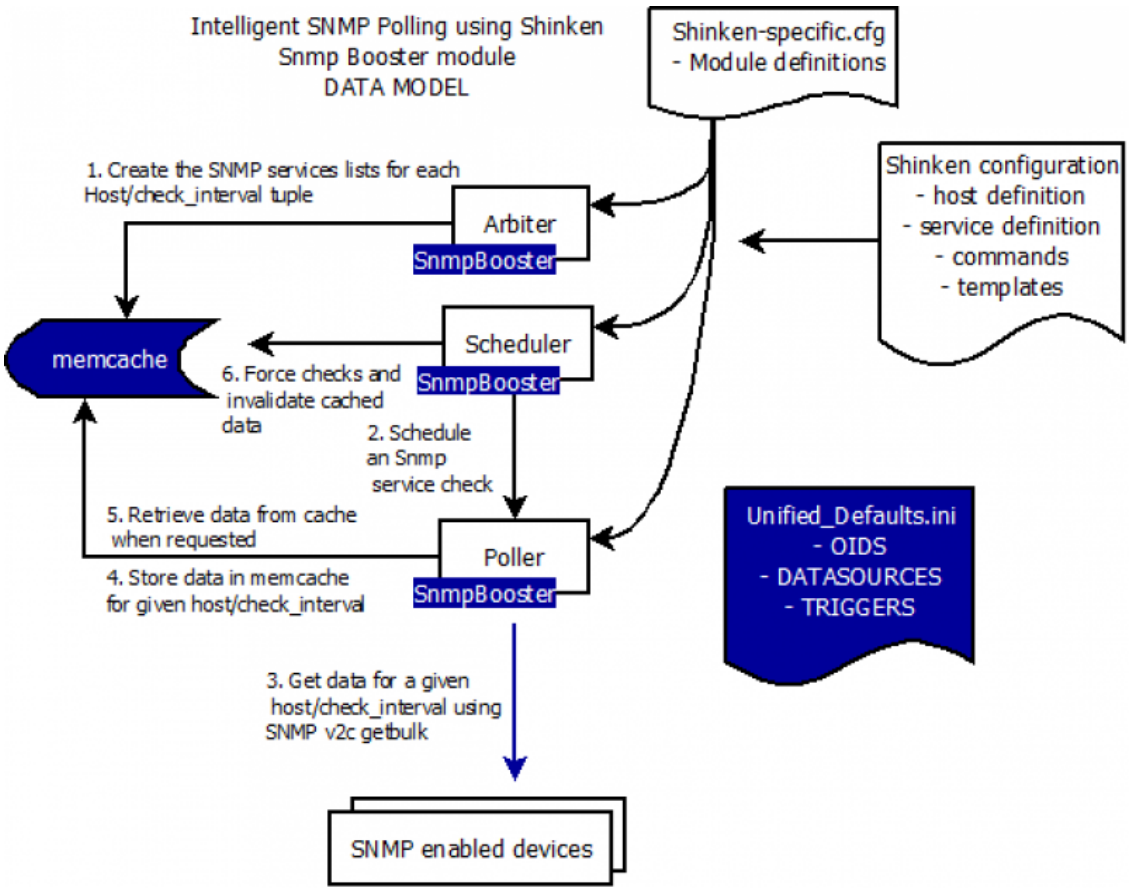

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

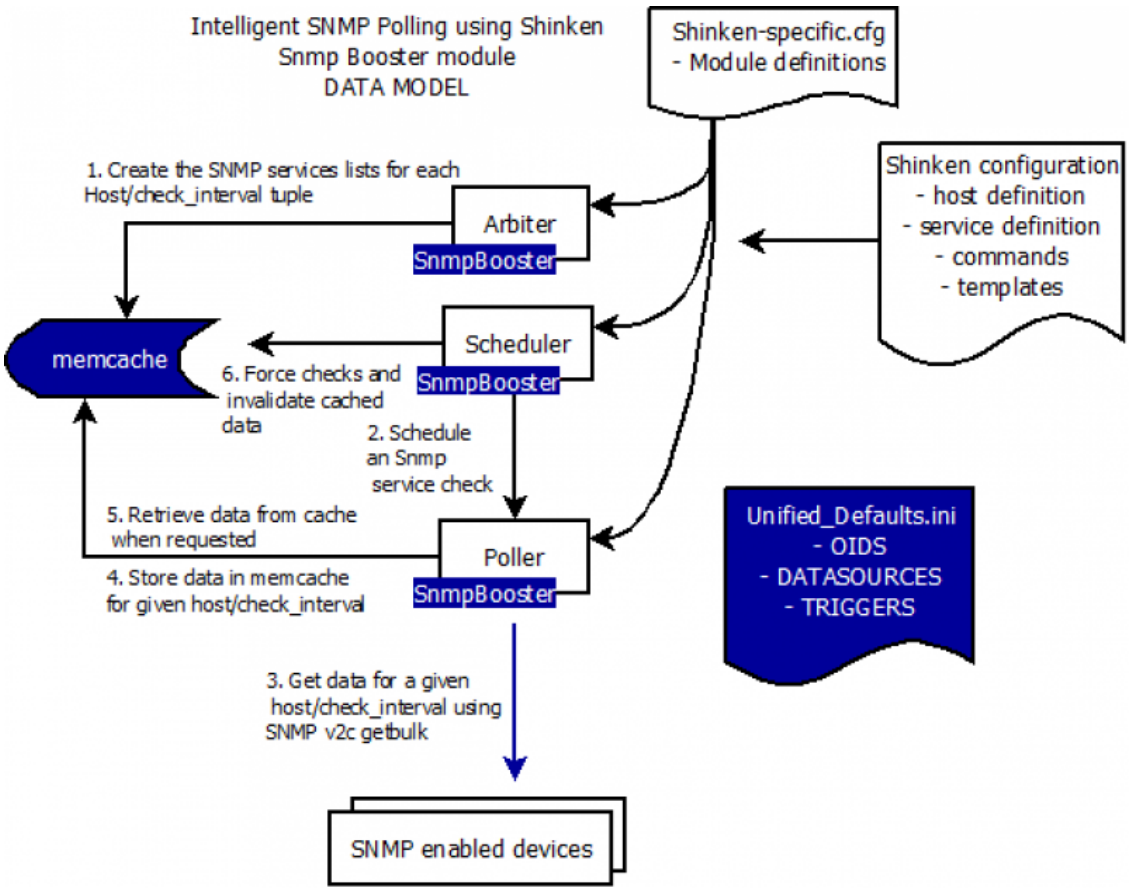

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

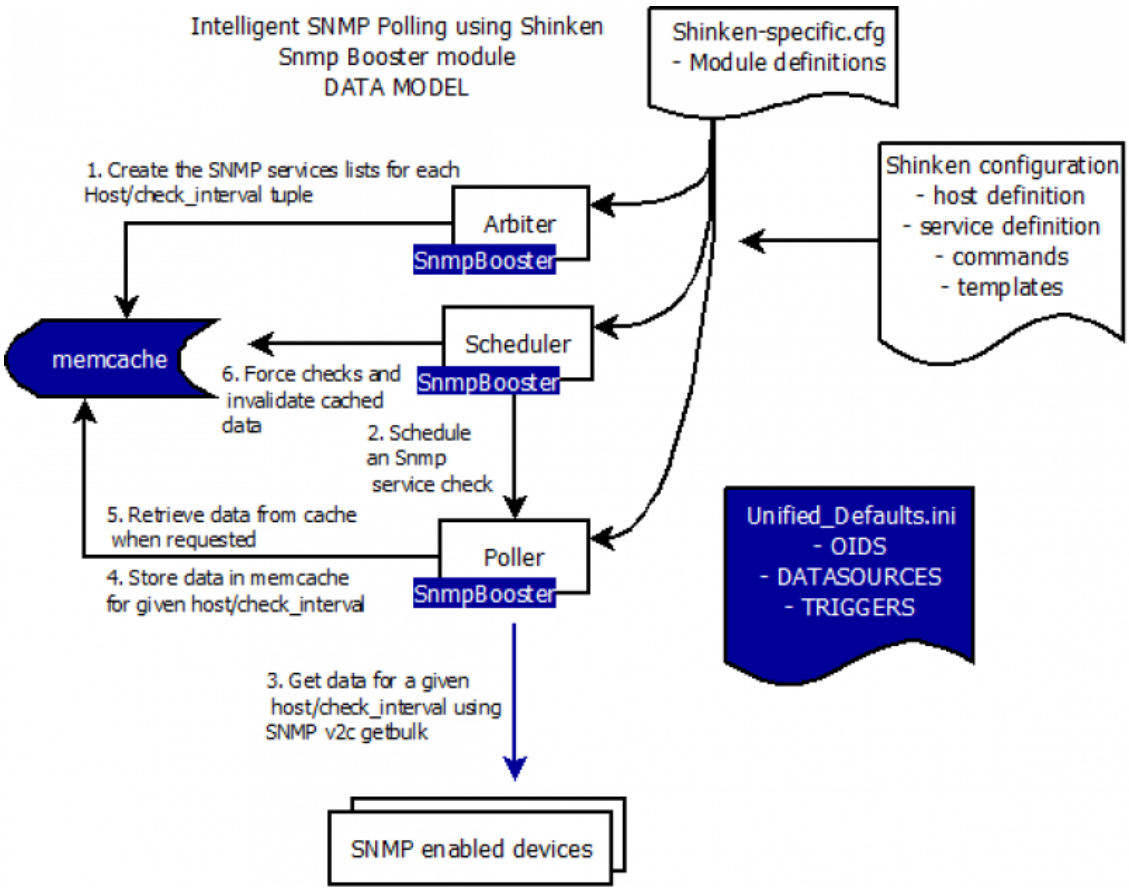

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

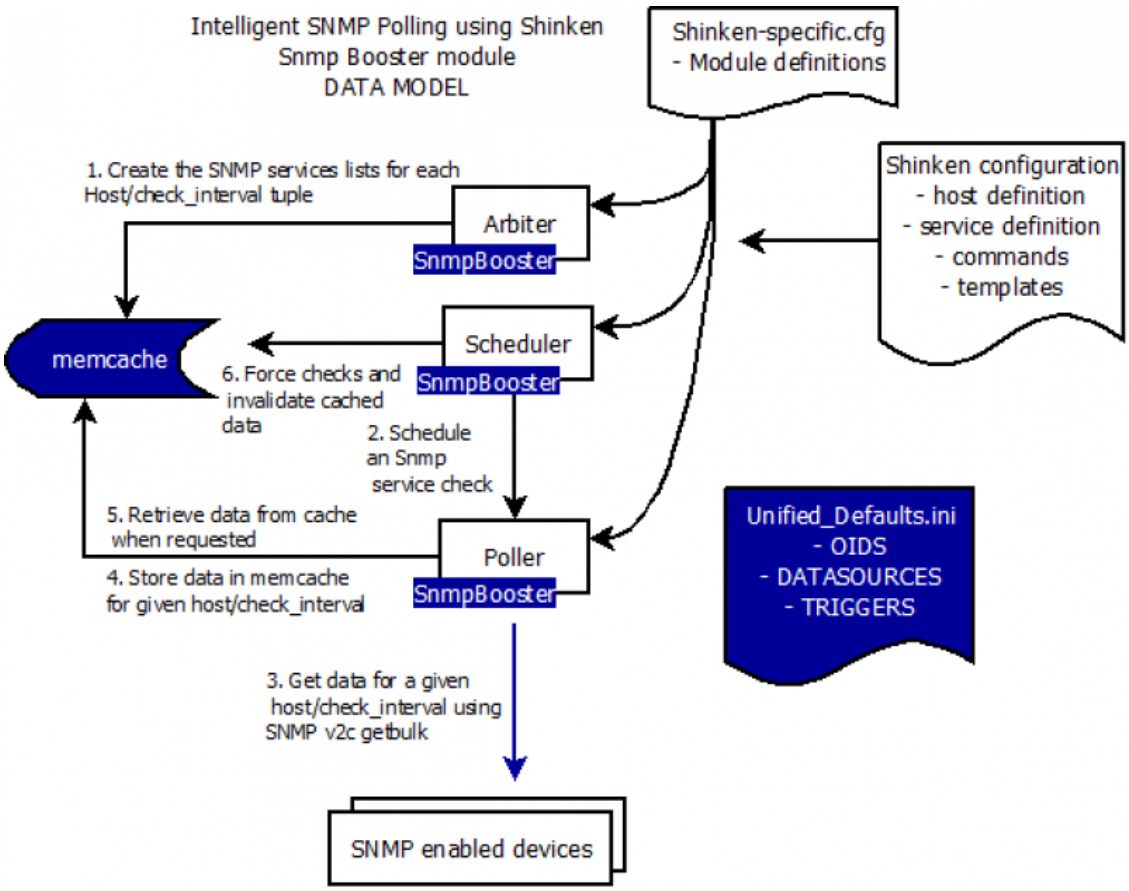

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

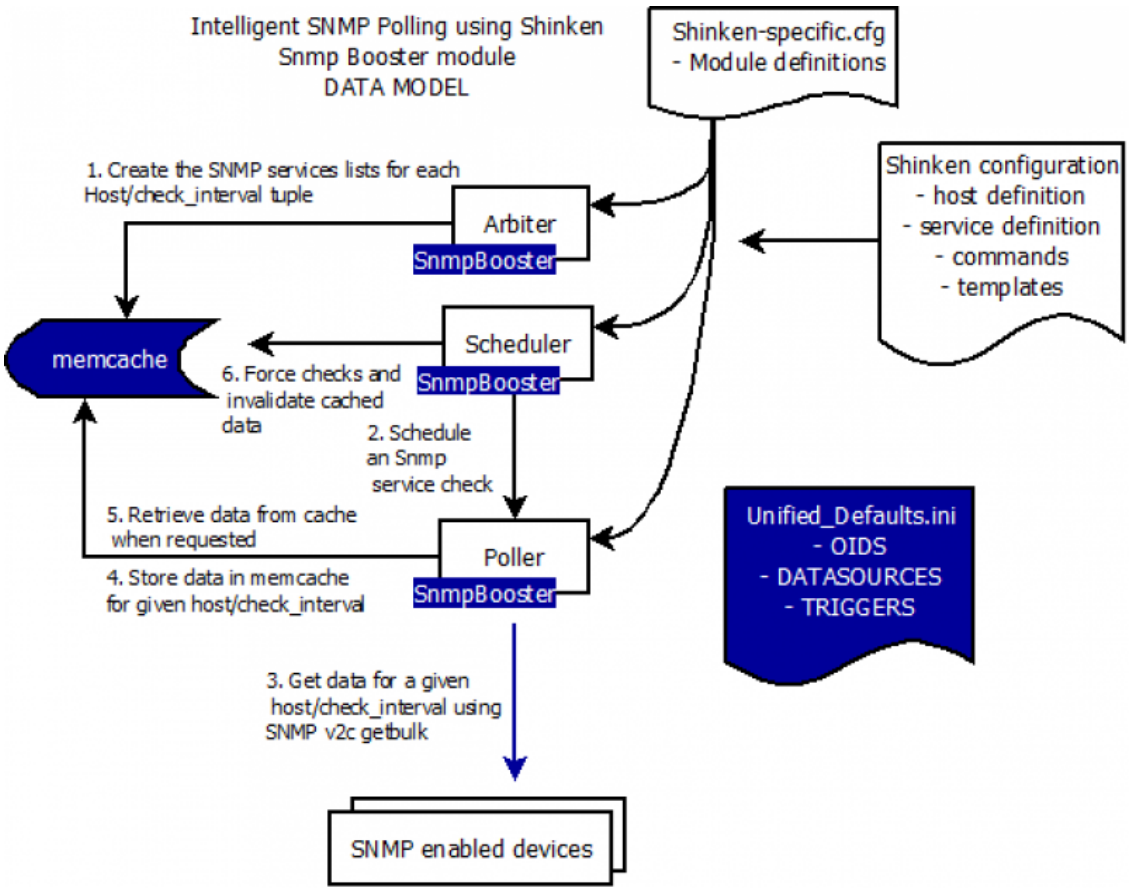

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

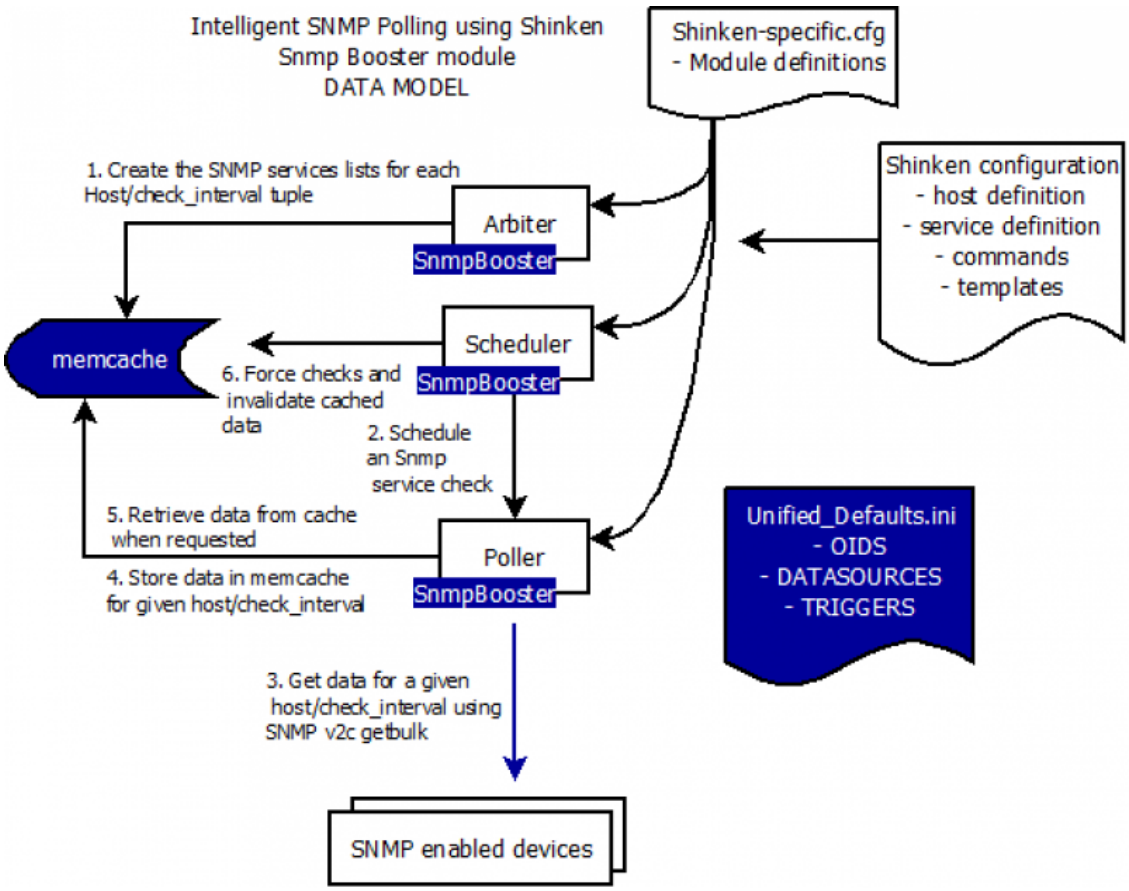

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

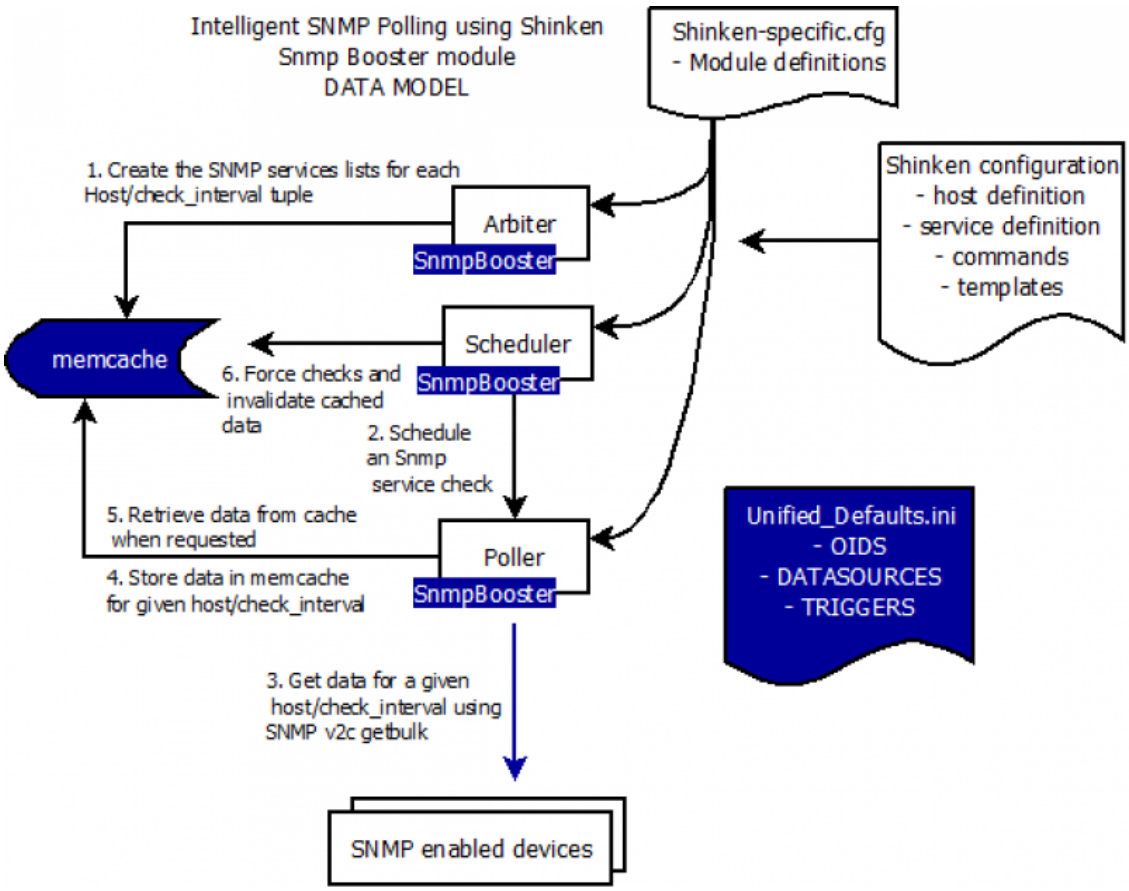

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

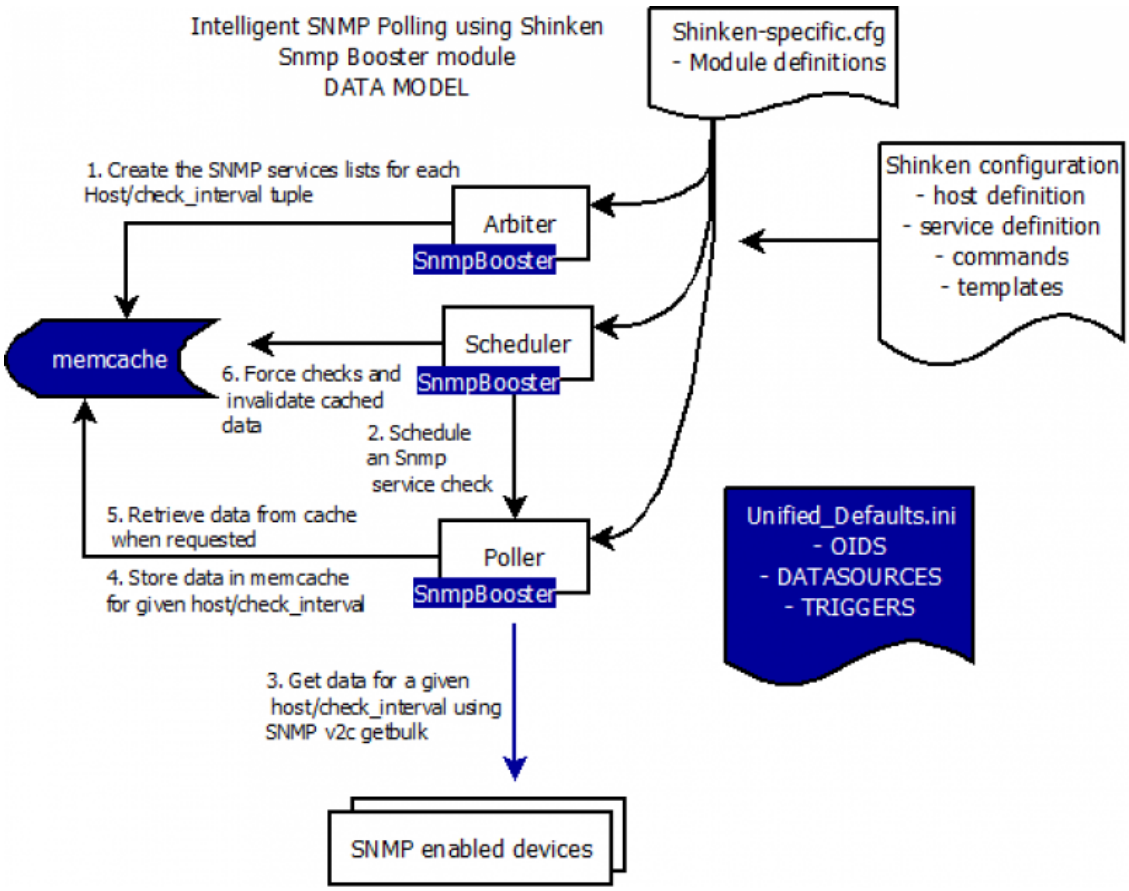

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

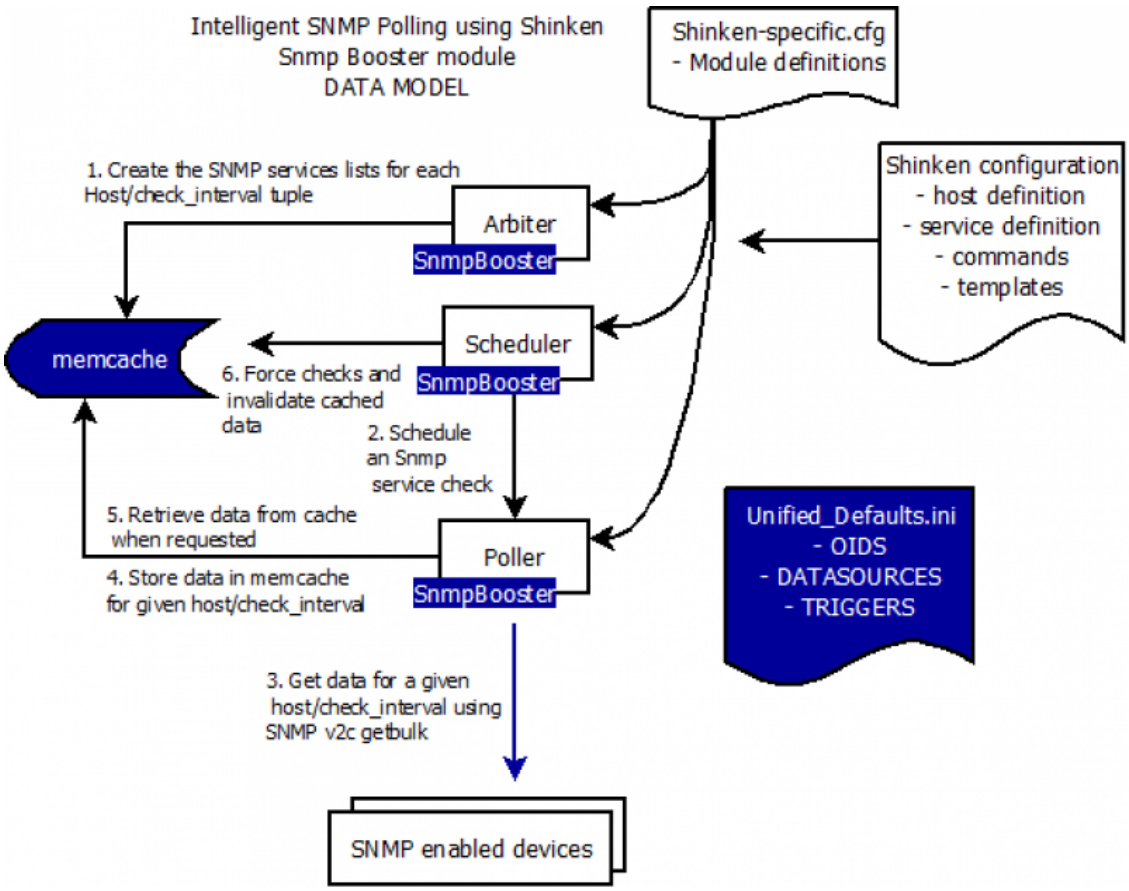

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

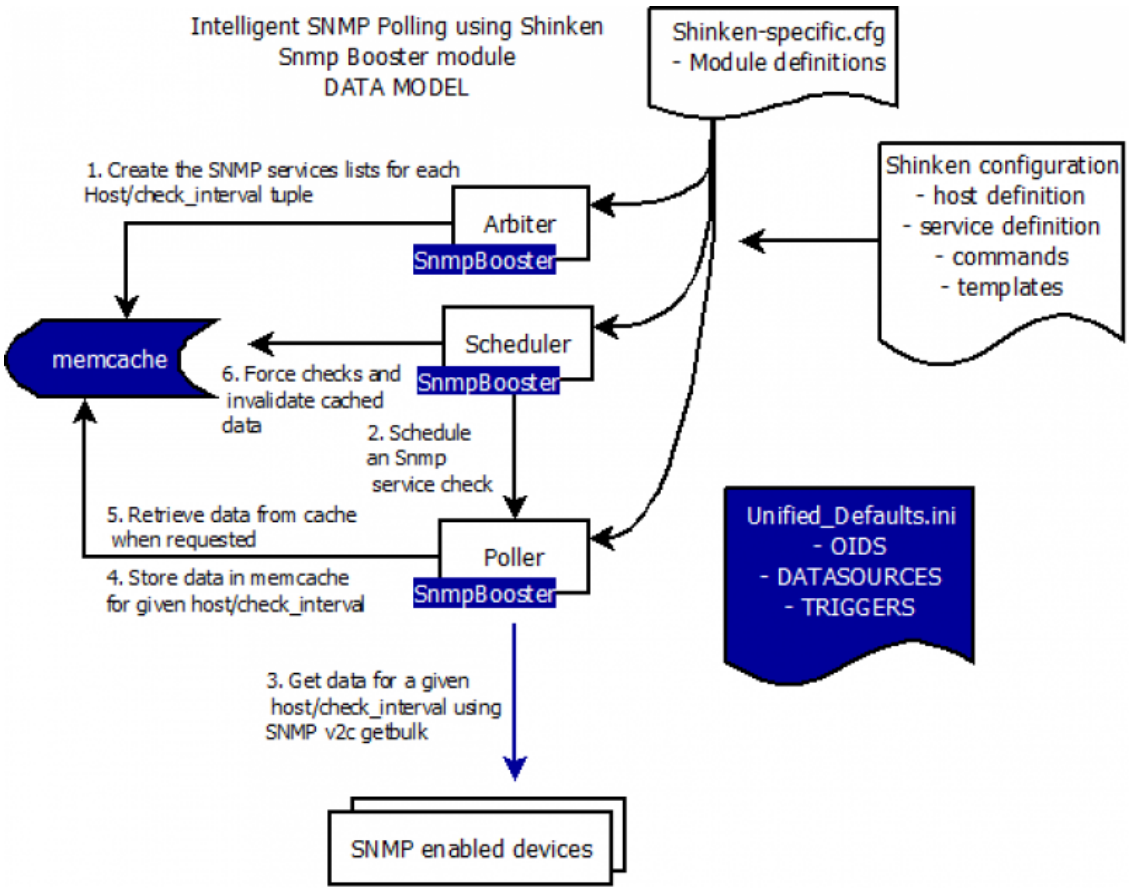

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

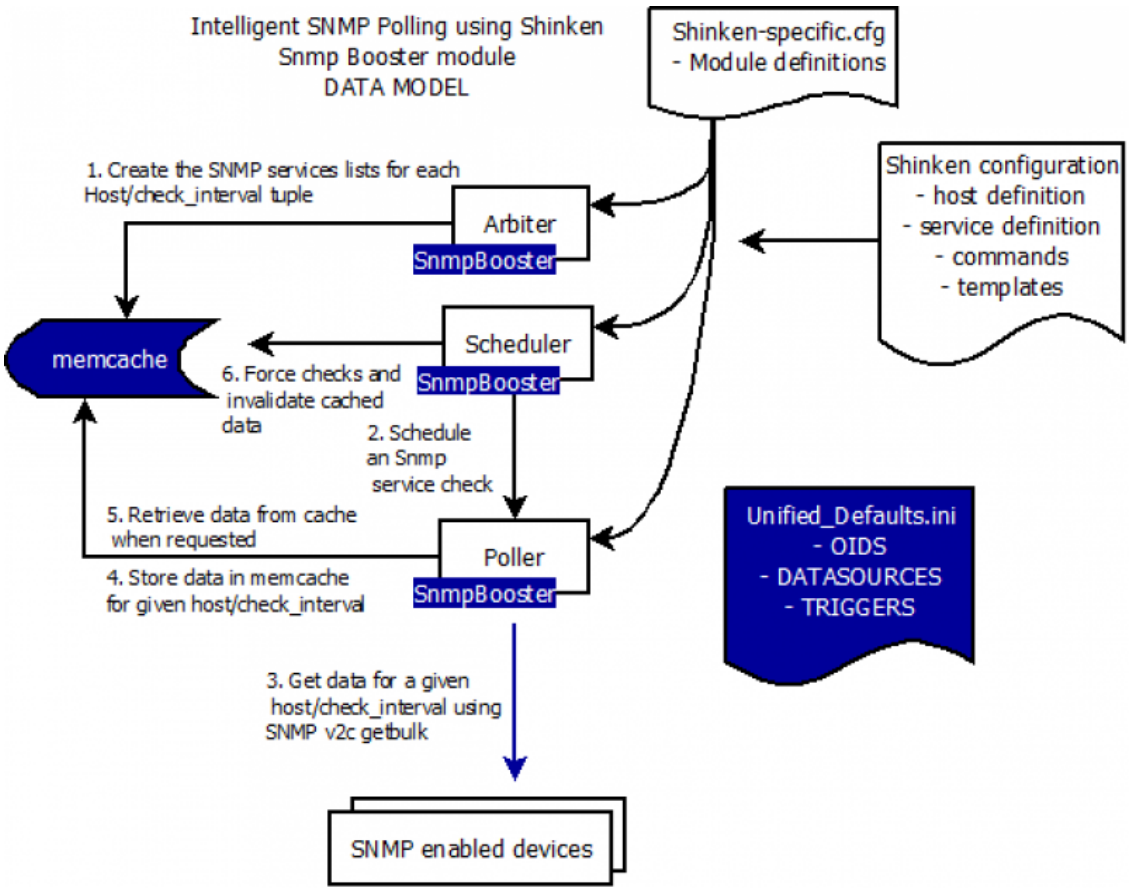

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

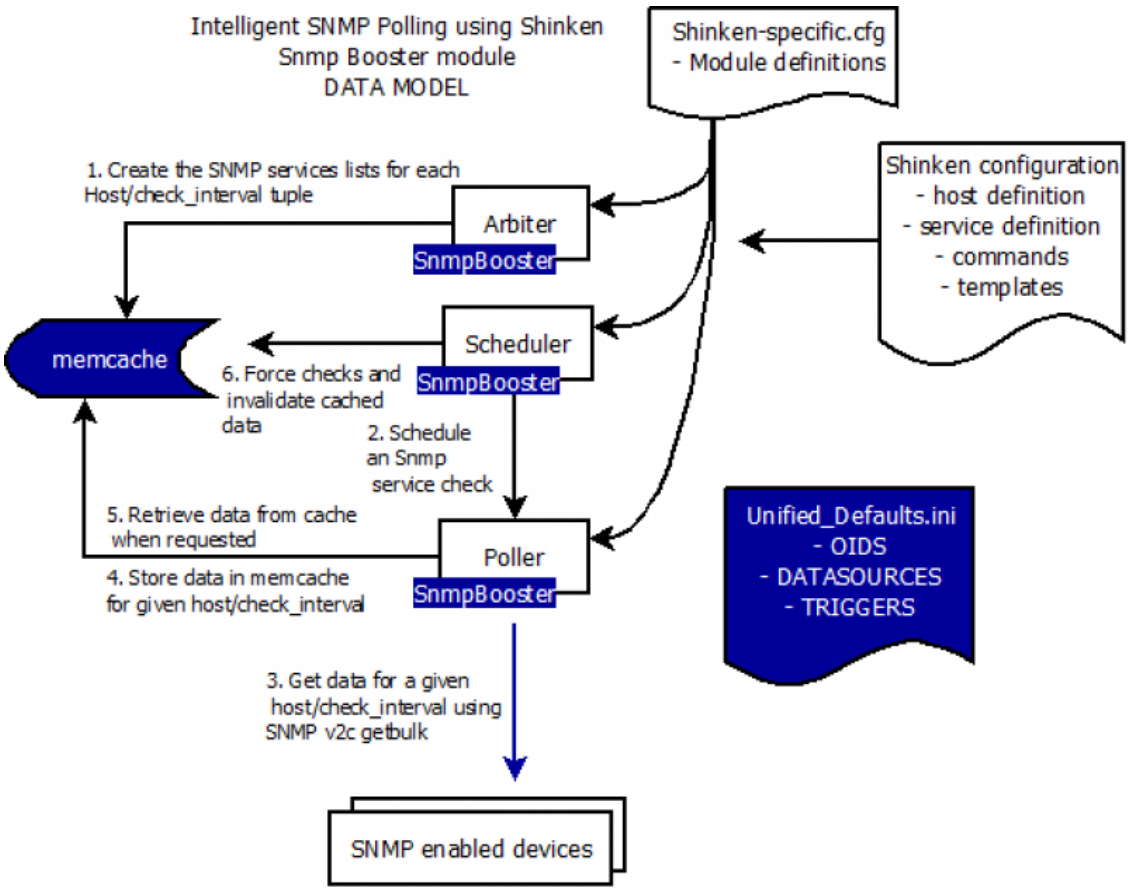

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

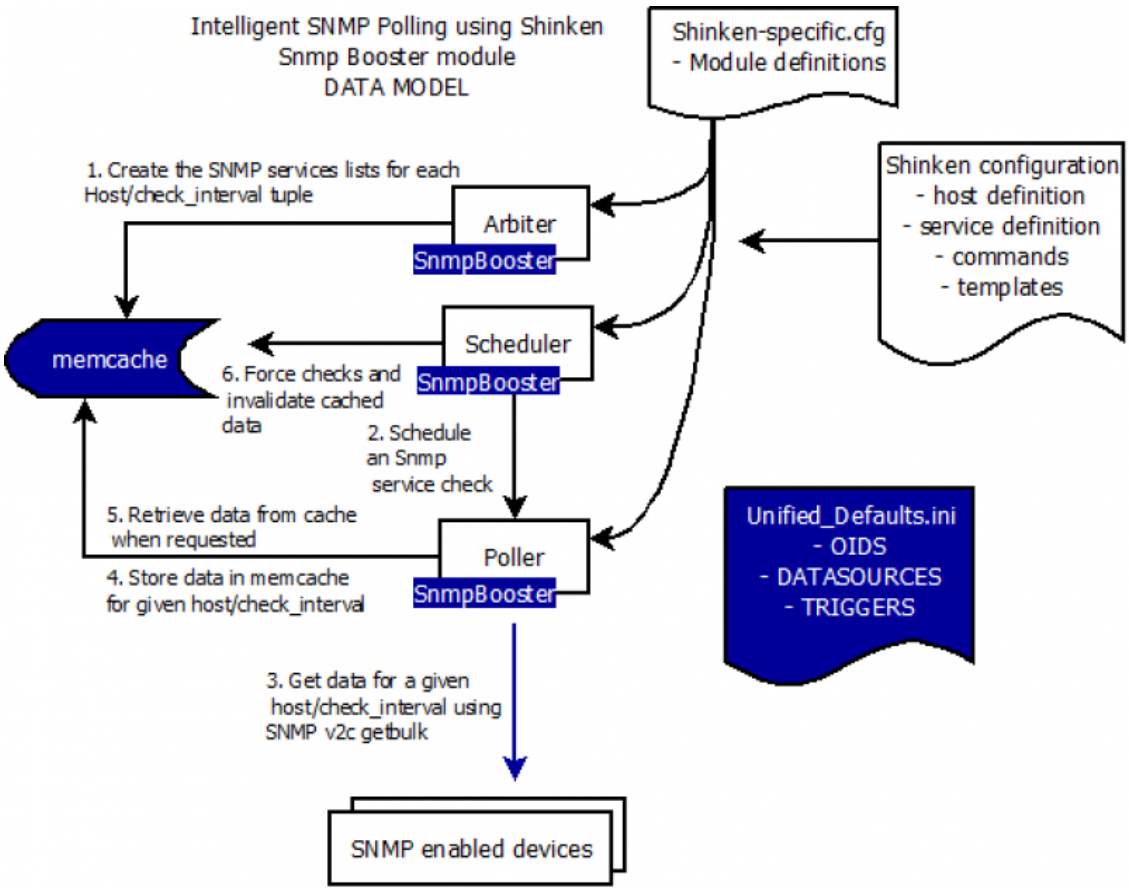

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

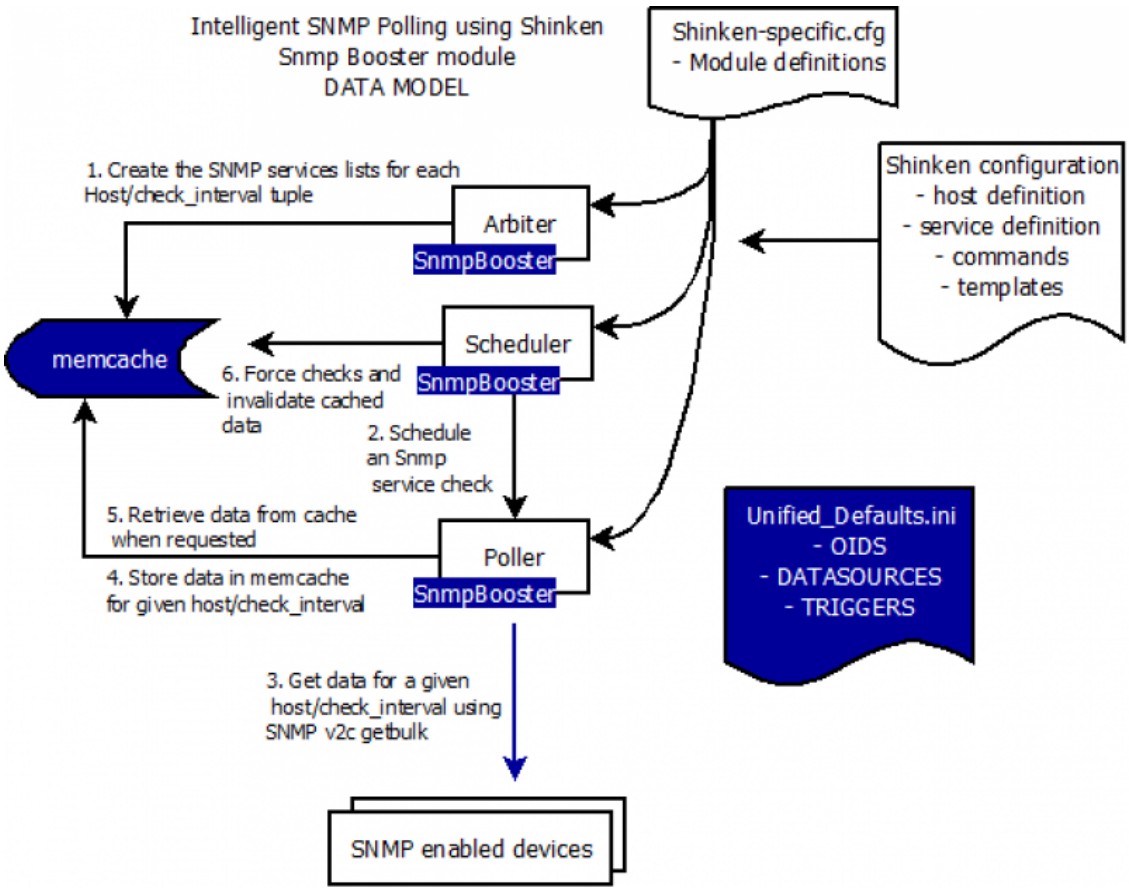

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

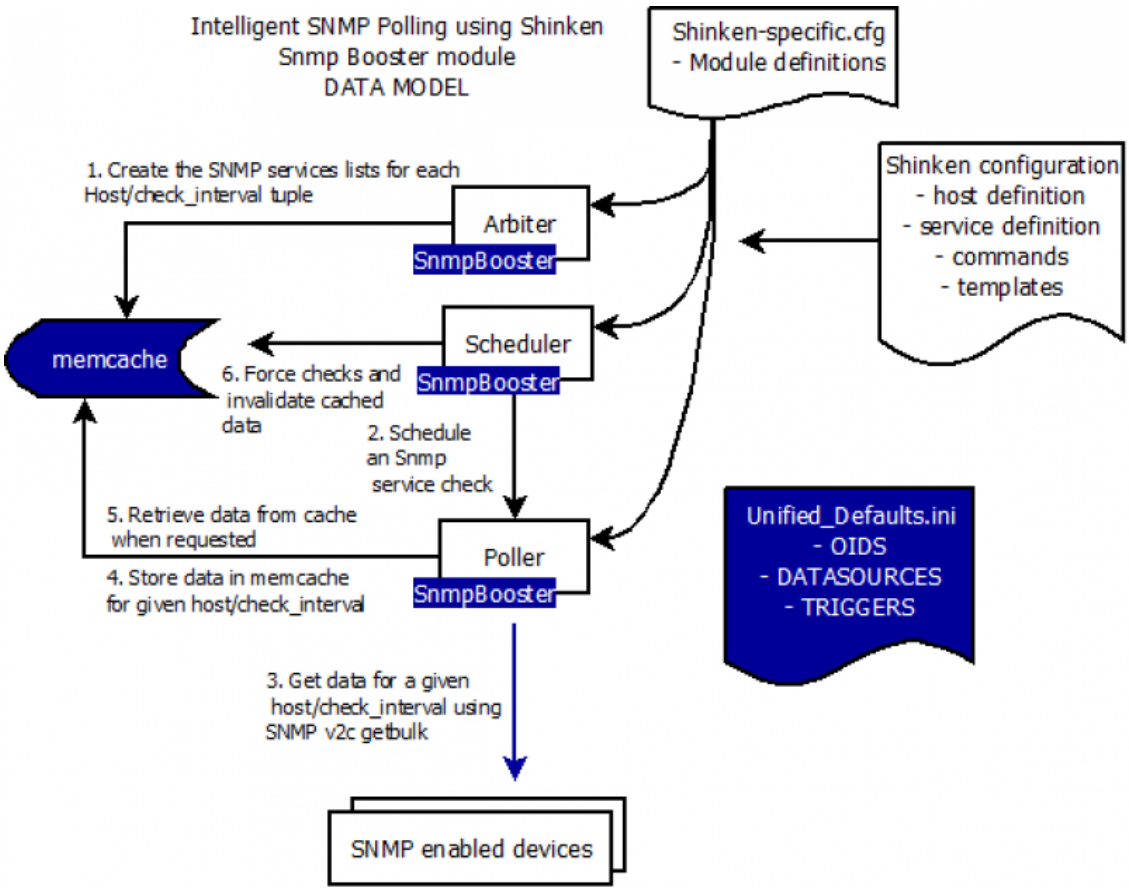

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

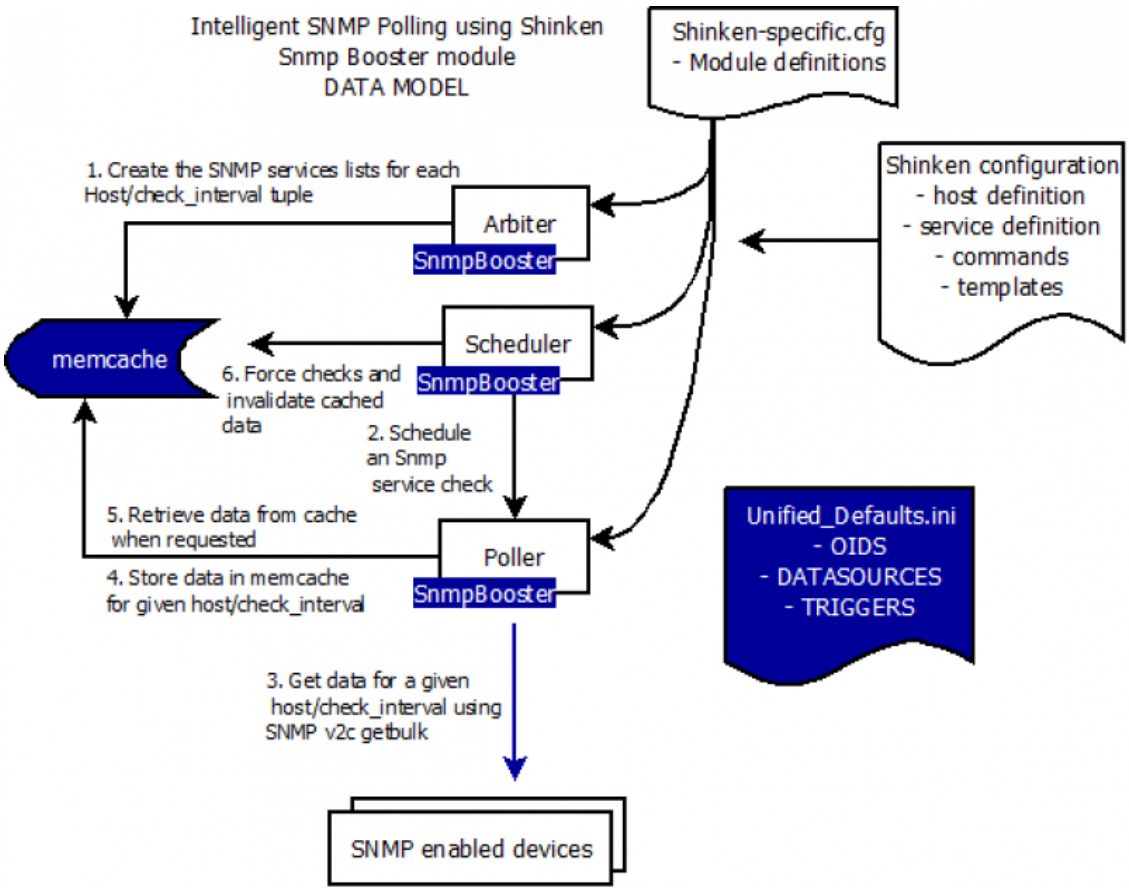

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

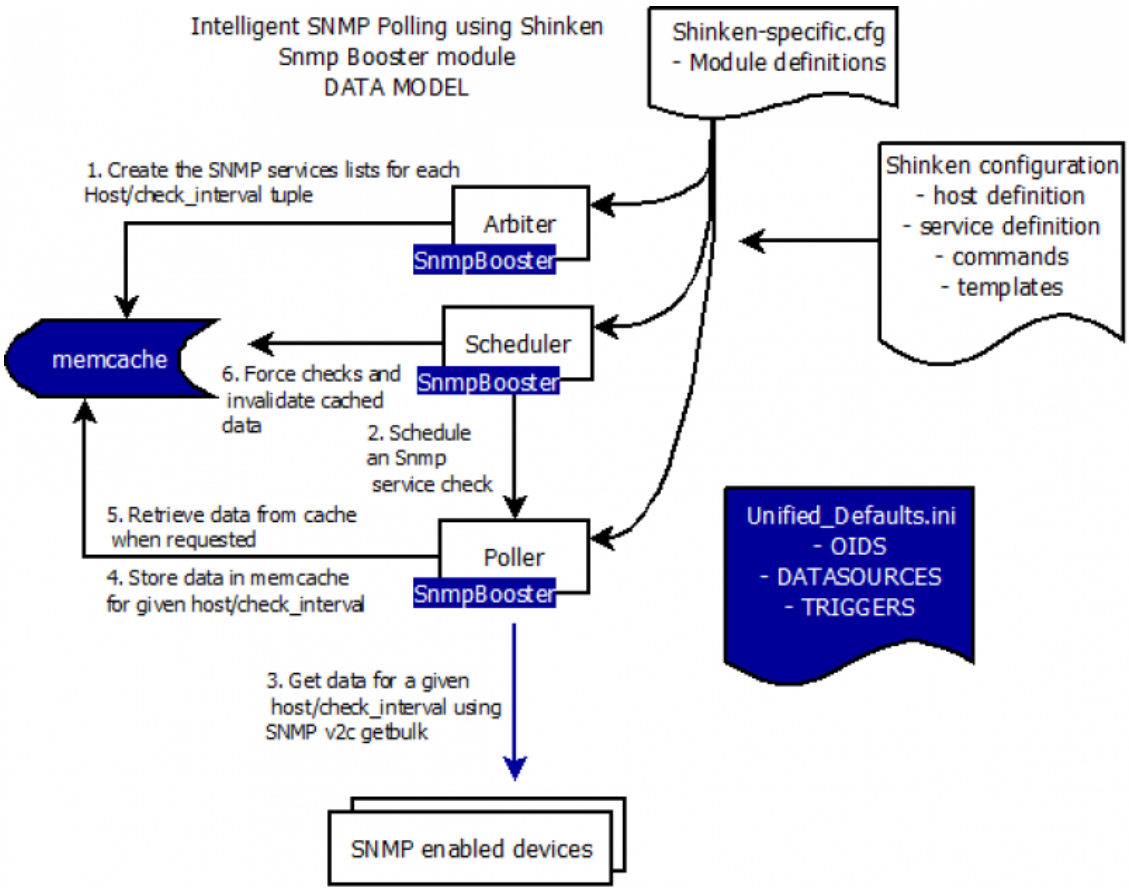

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

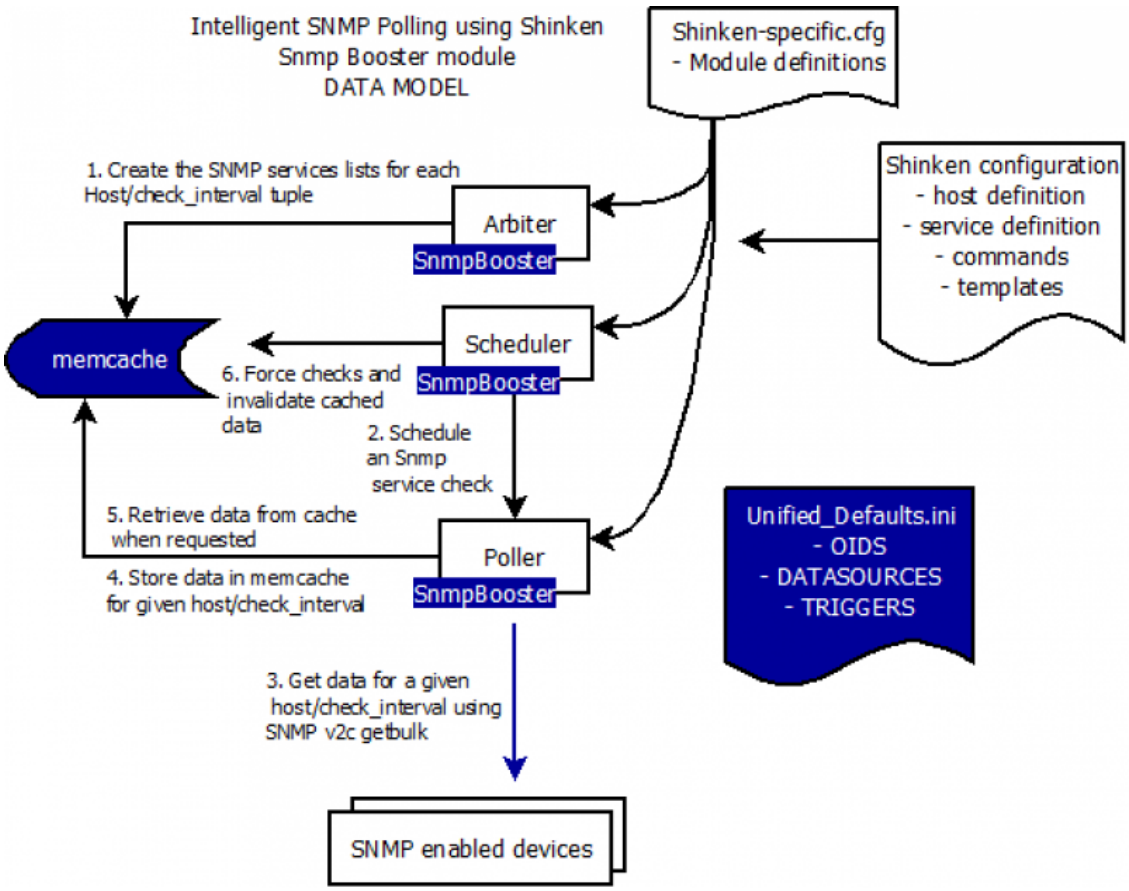

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

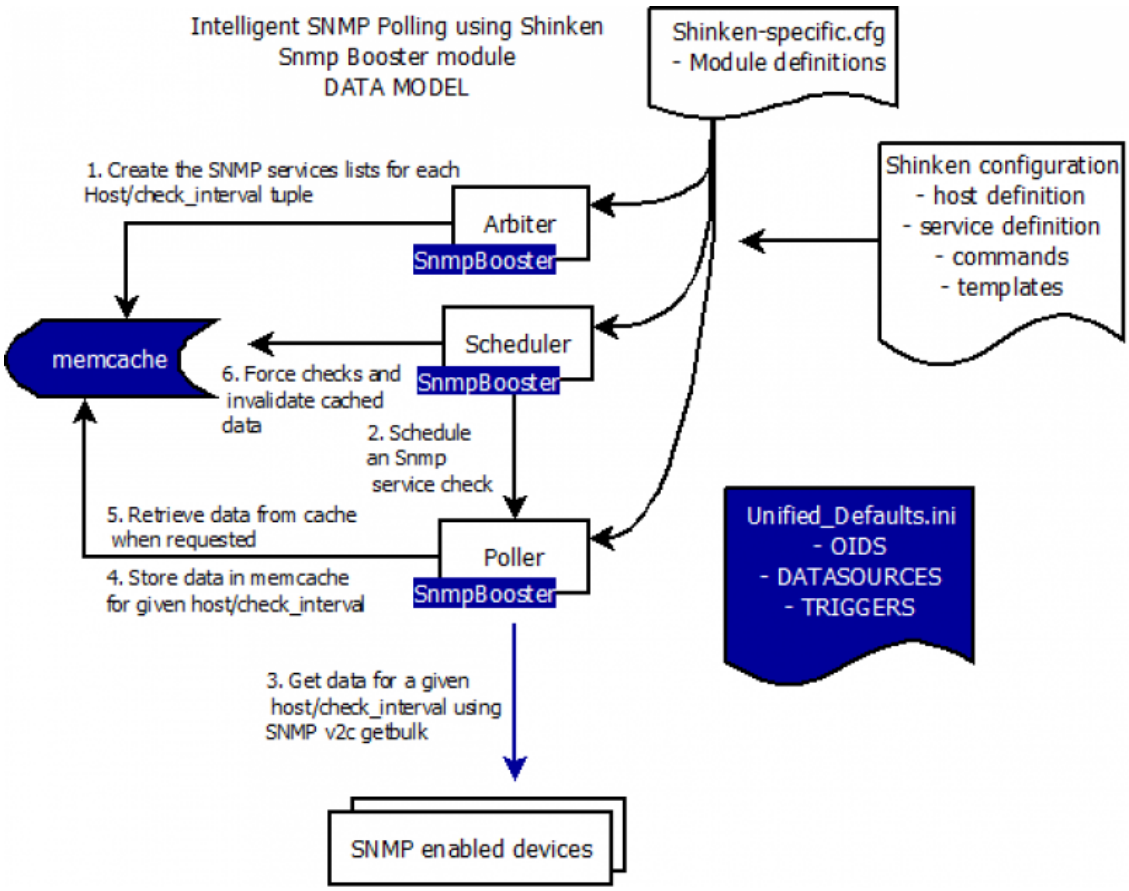

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

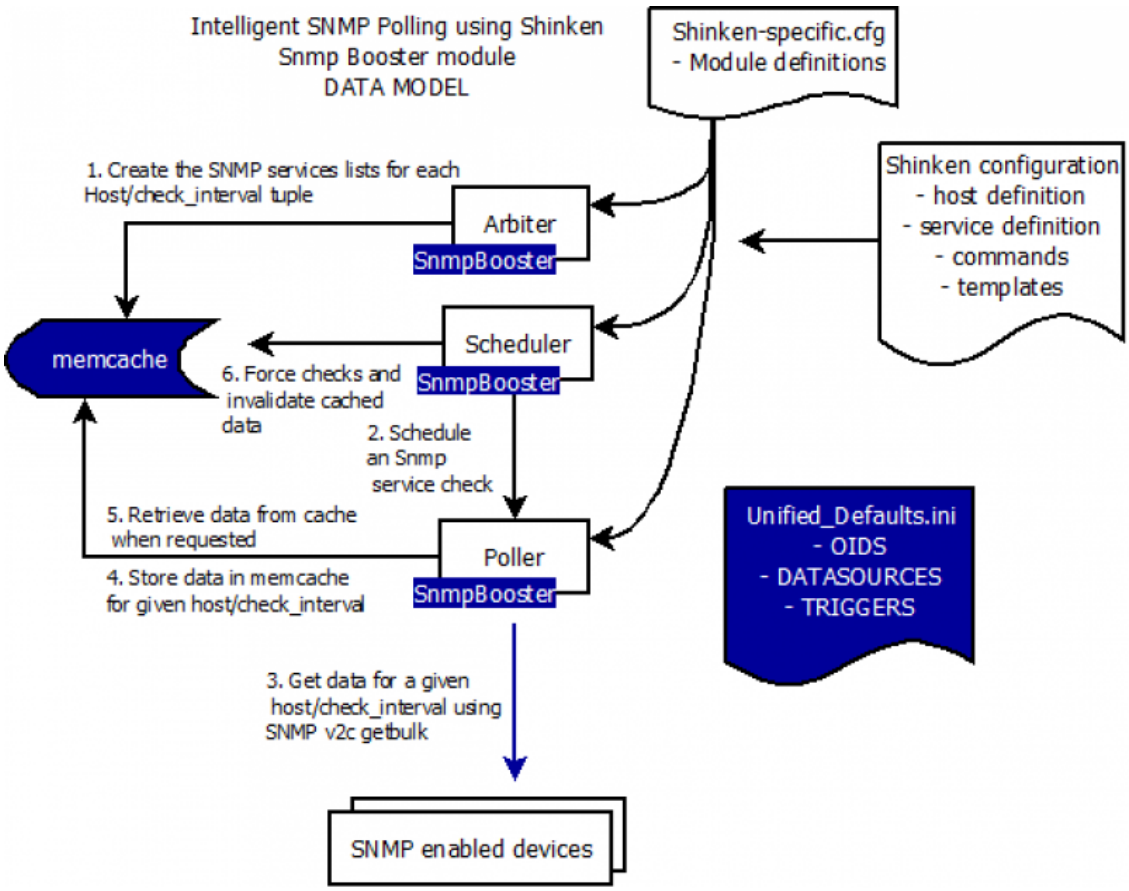

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

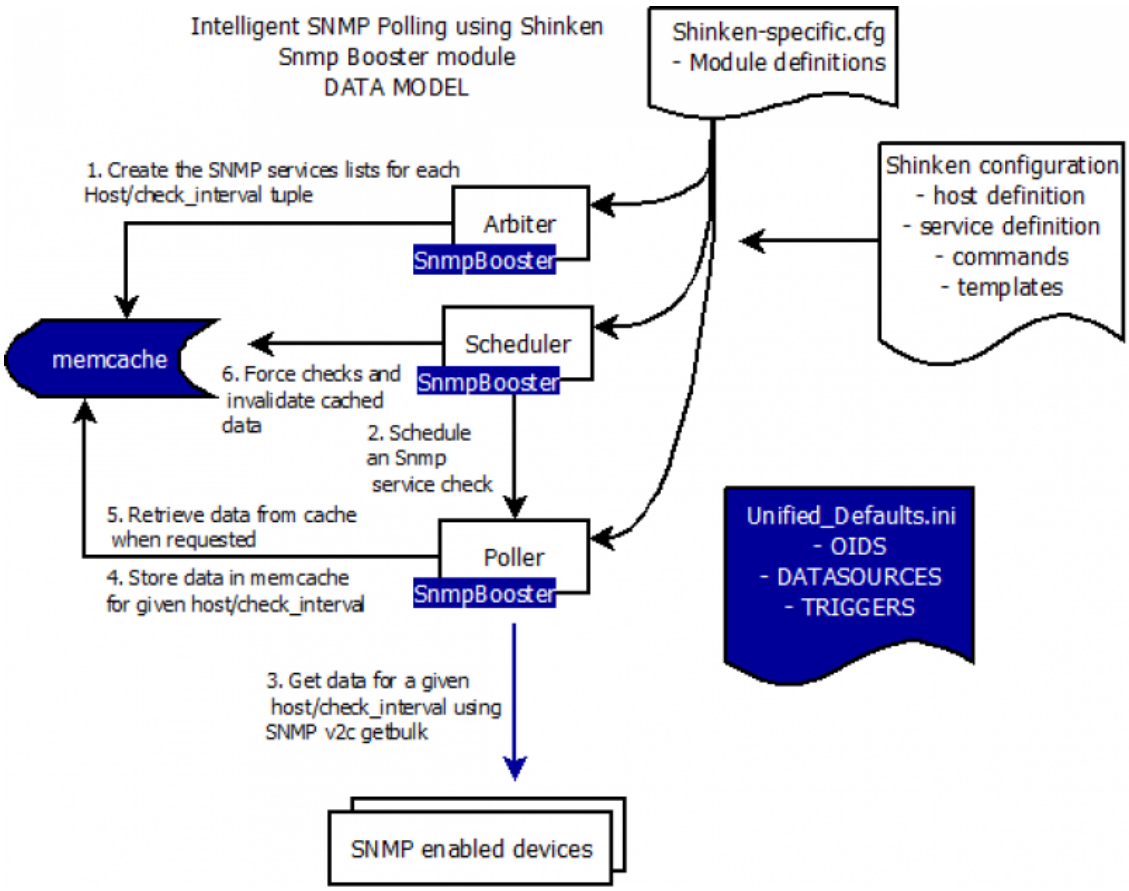

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

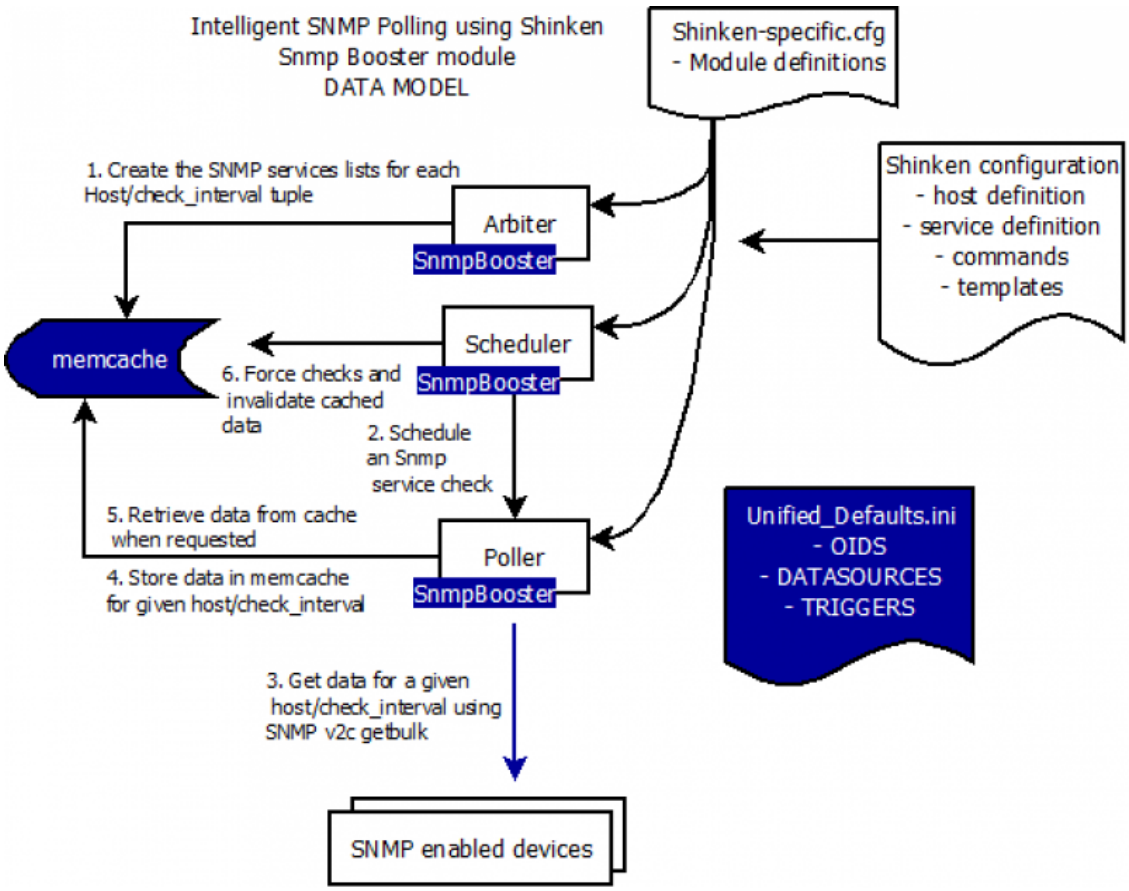

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

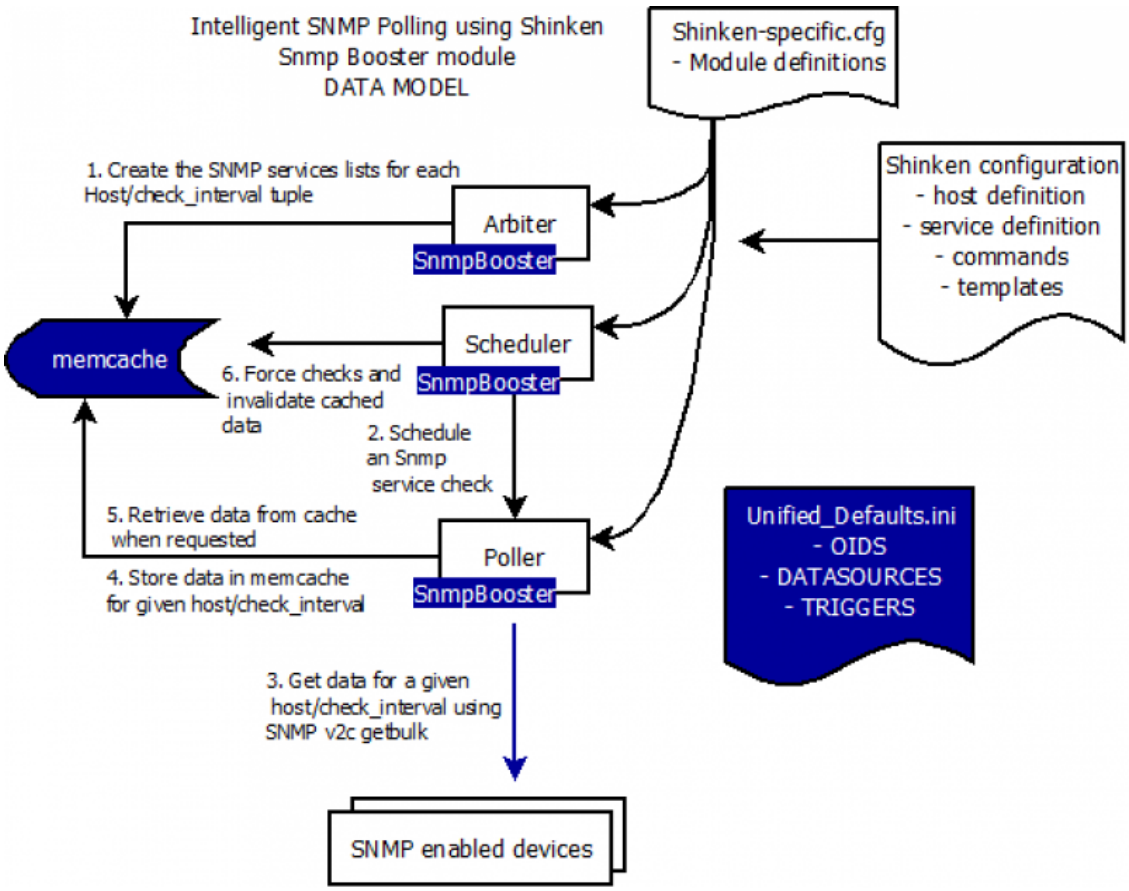

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

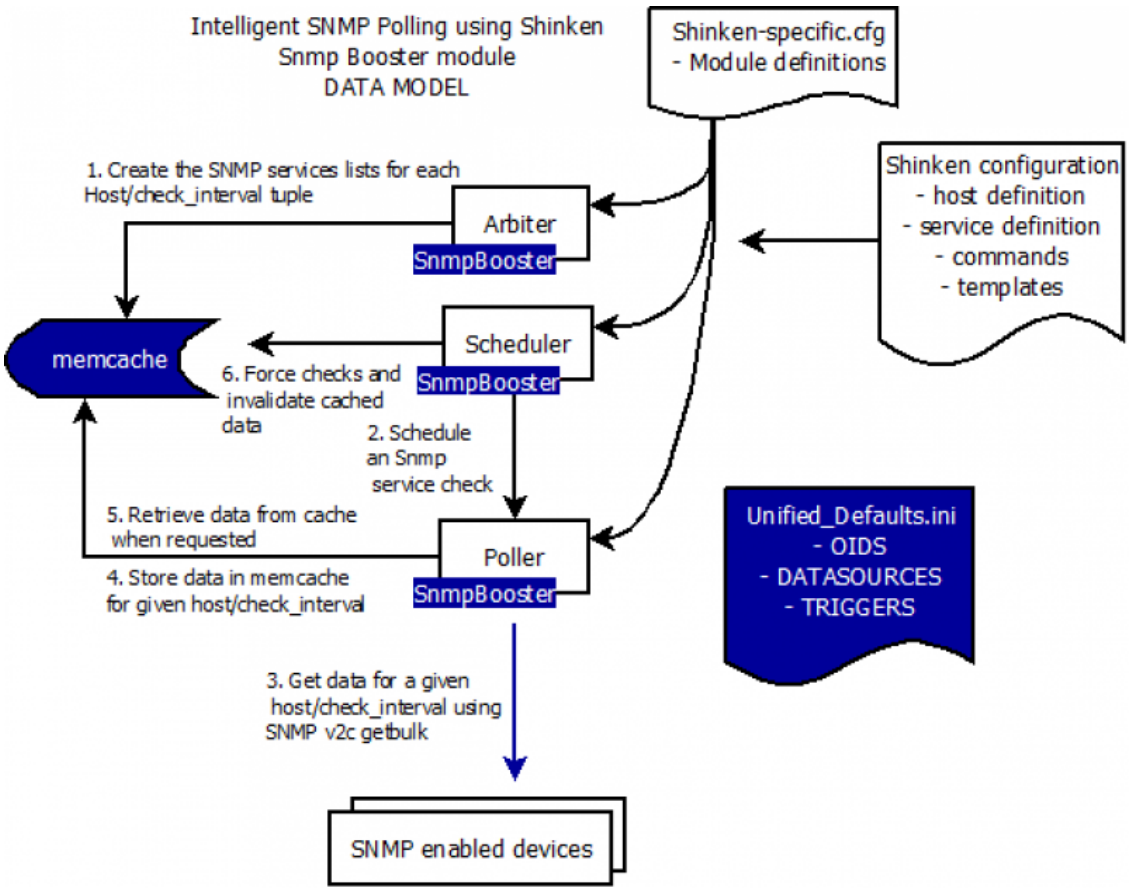

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

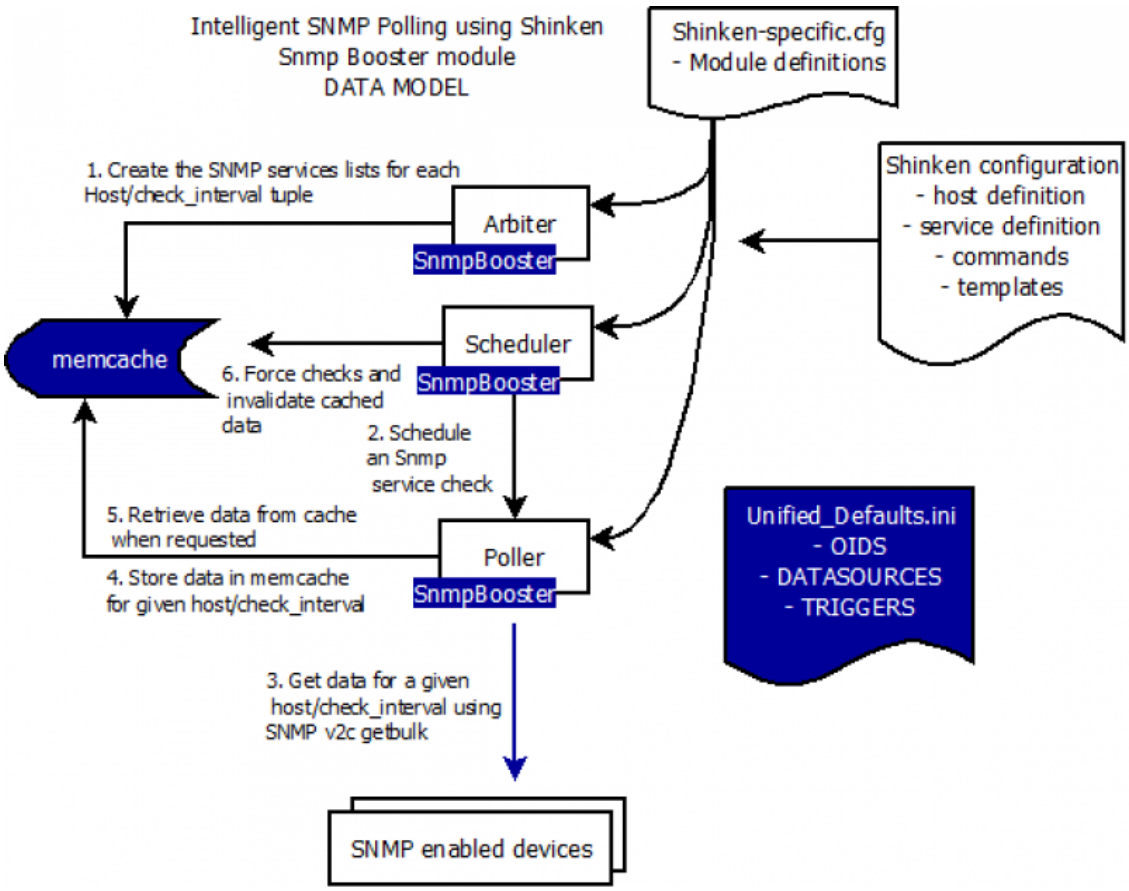

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

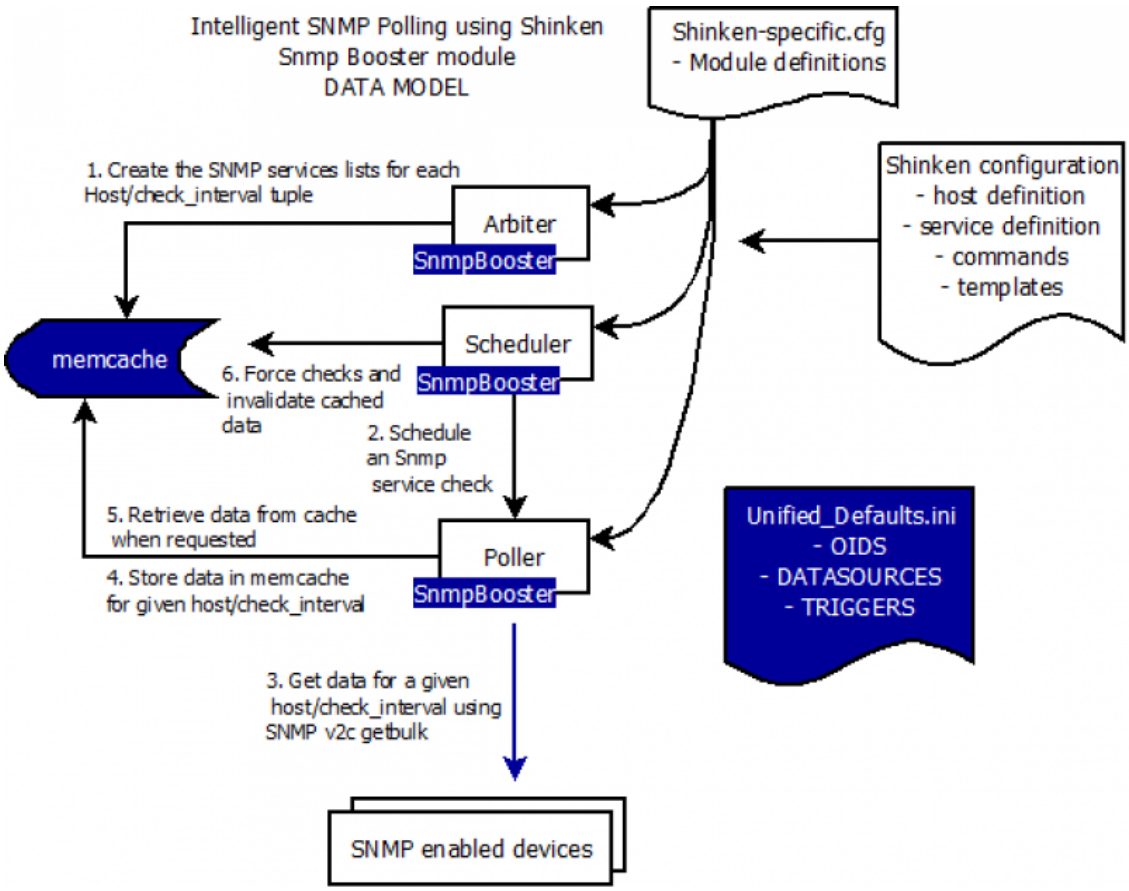

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

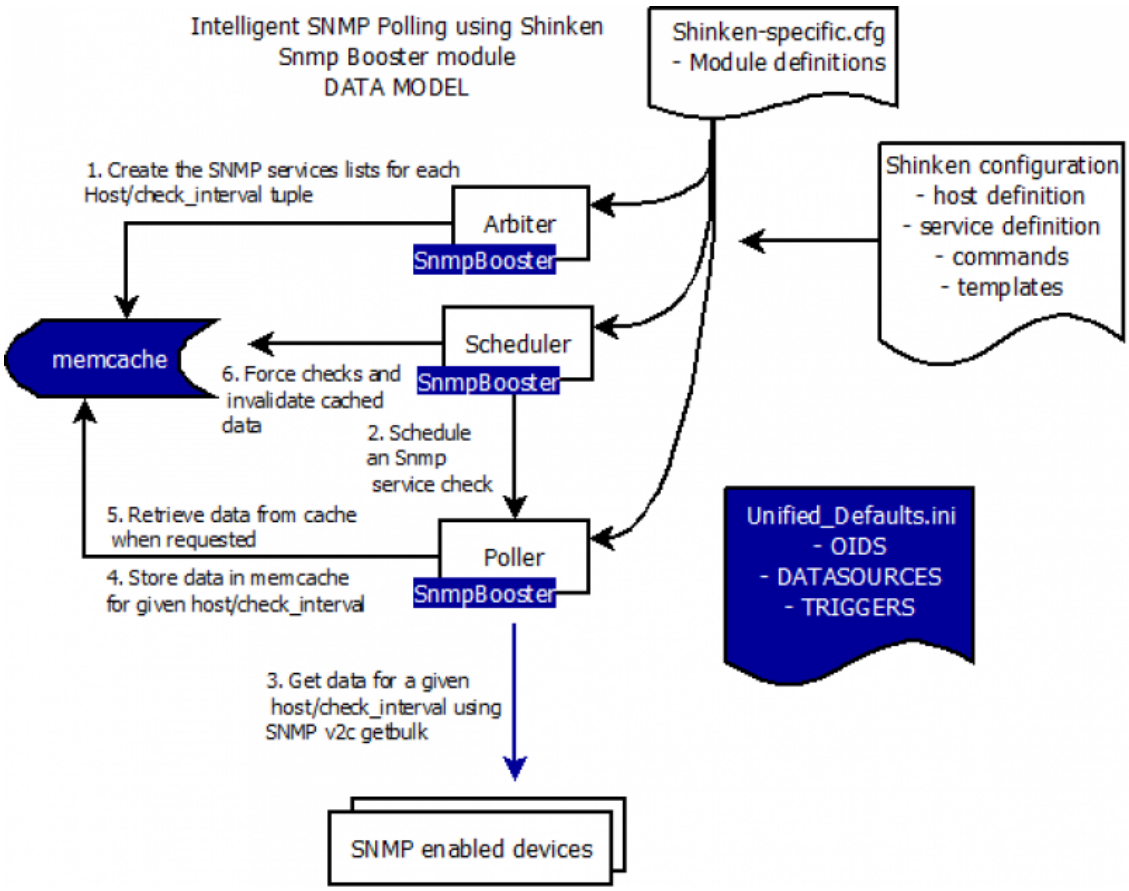

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

::

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

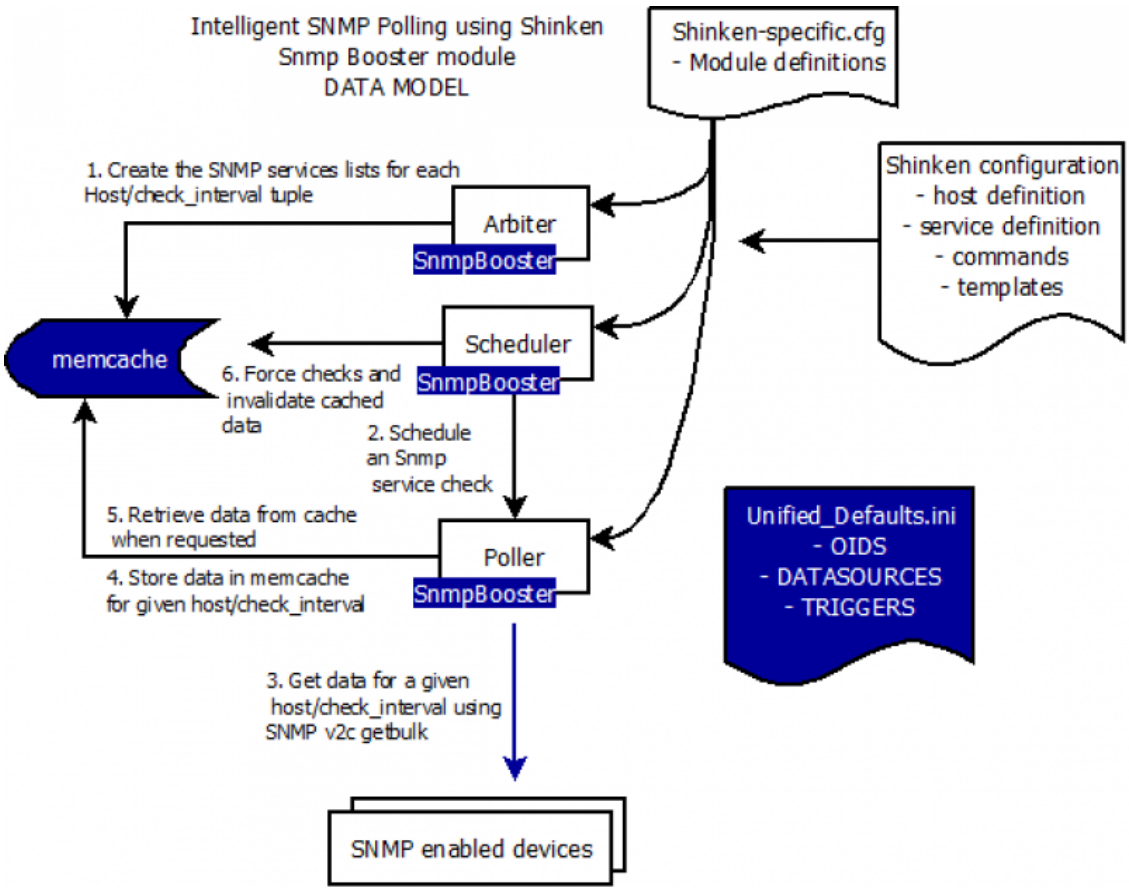

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

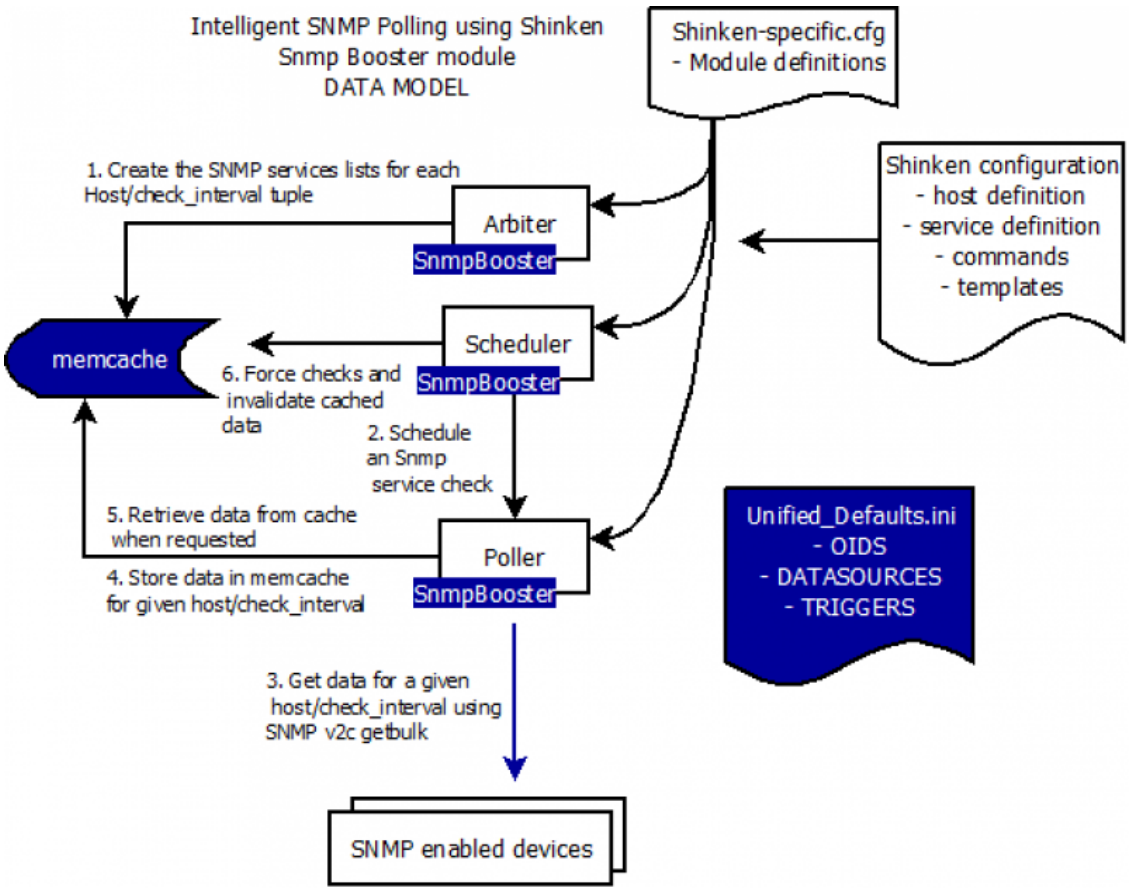

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

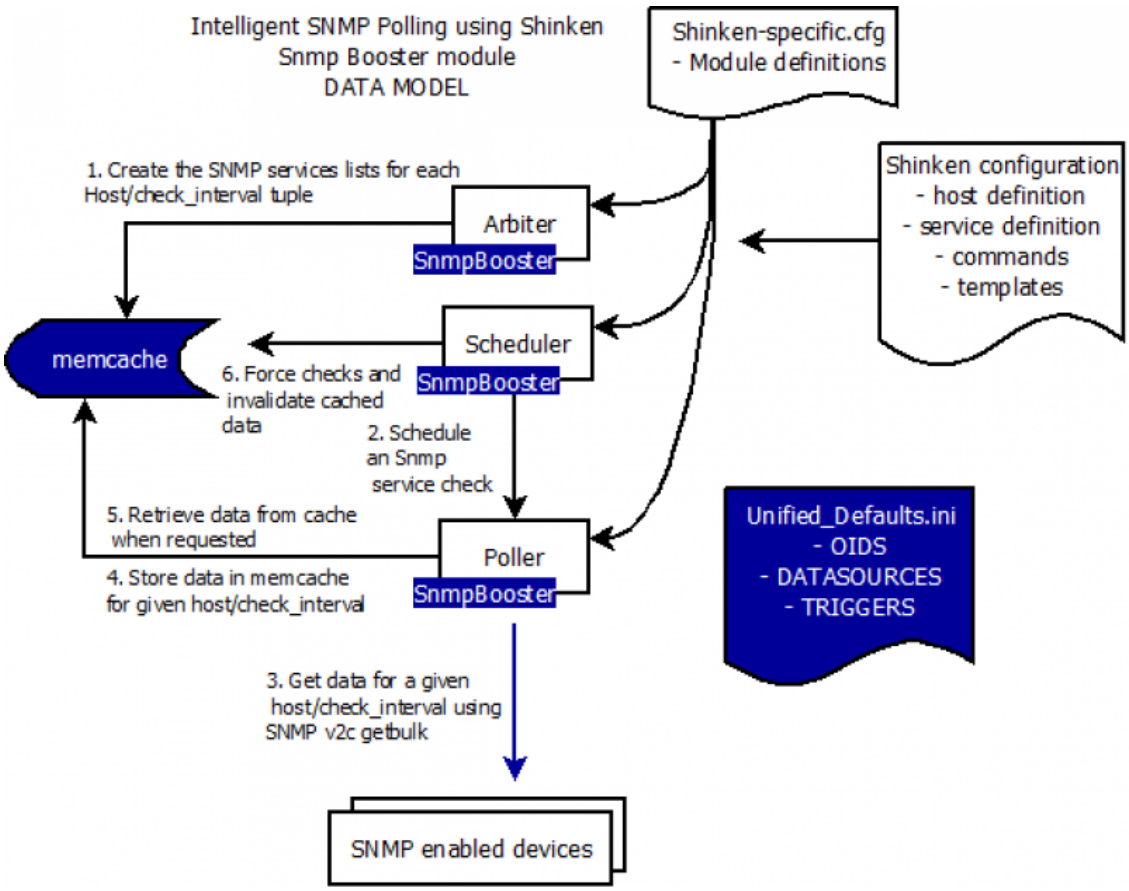

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

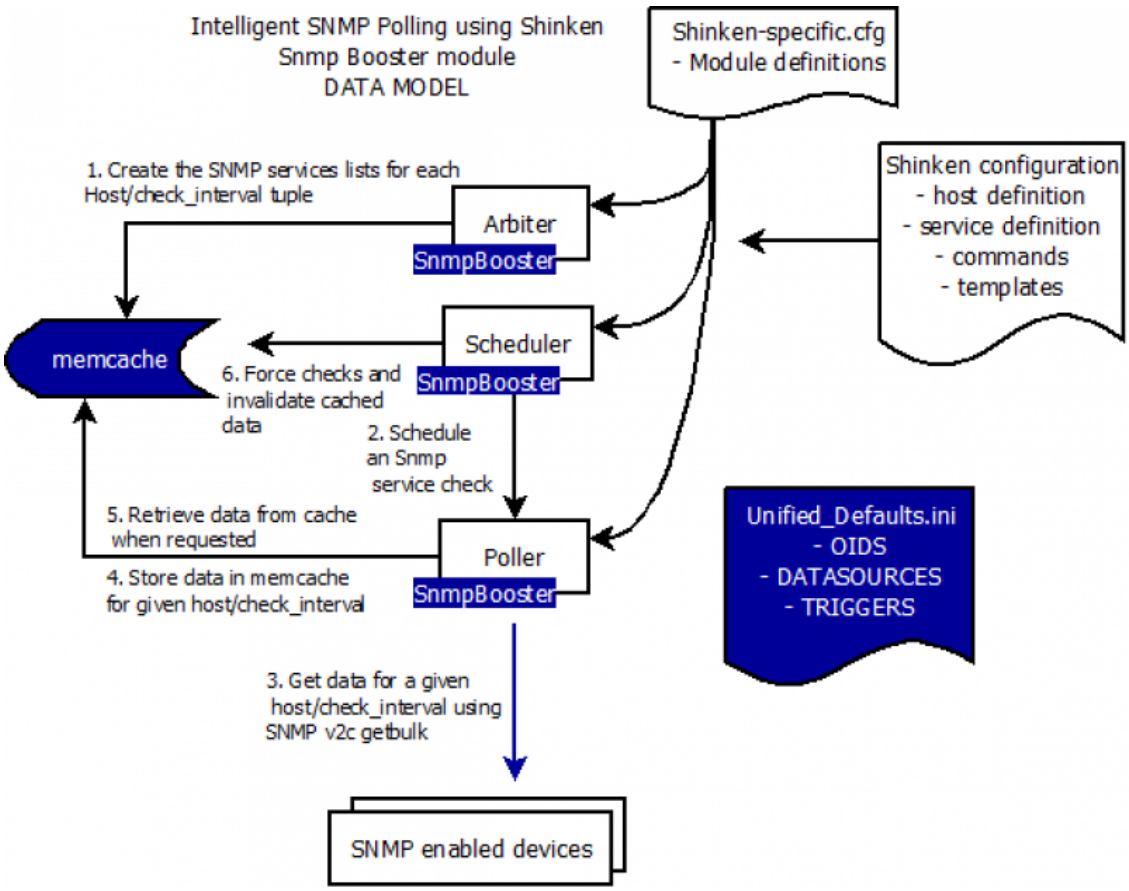

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

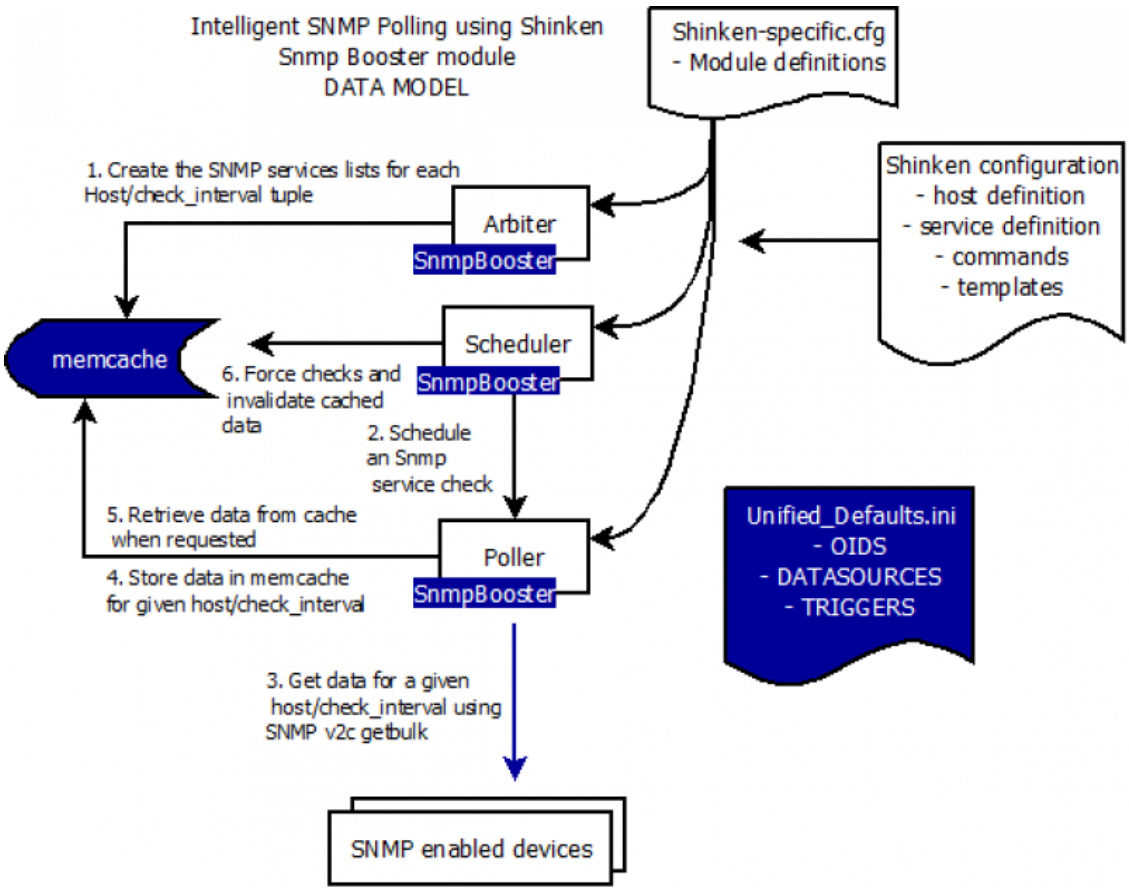

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

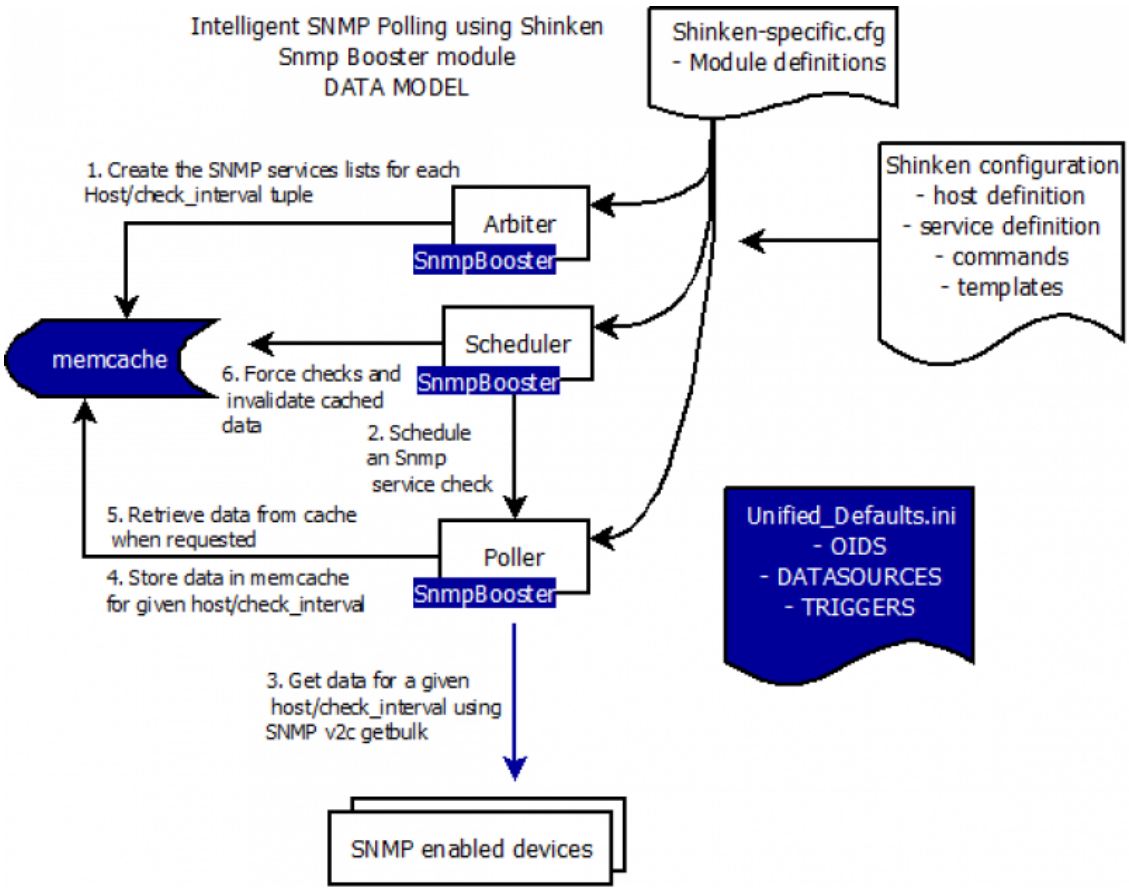

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

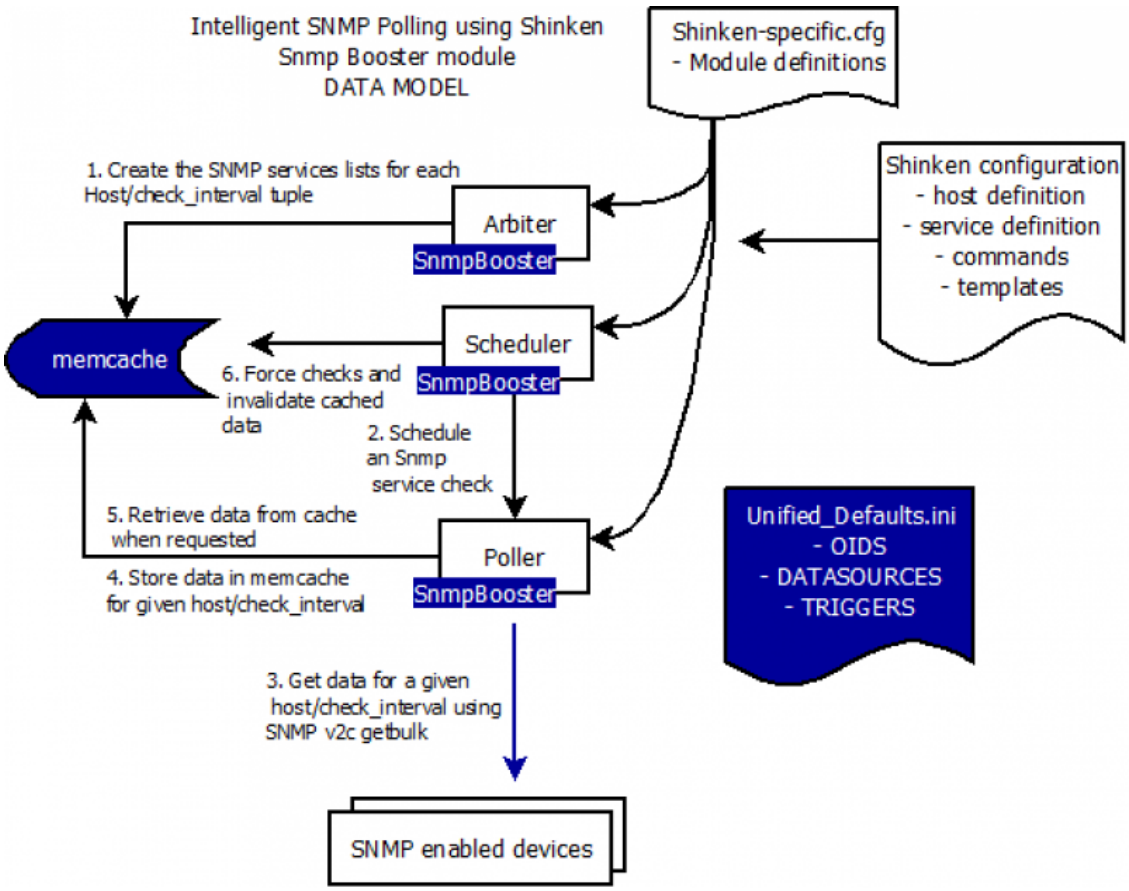

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

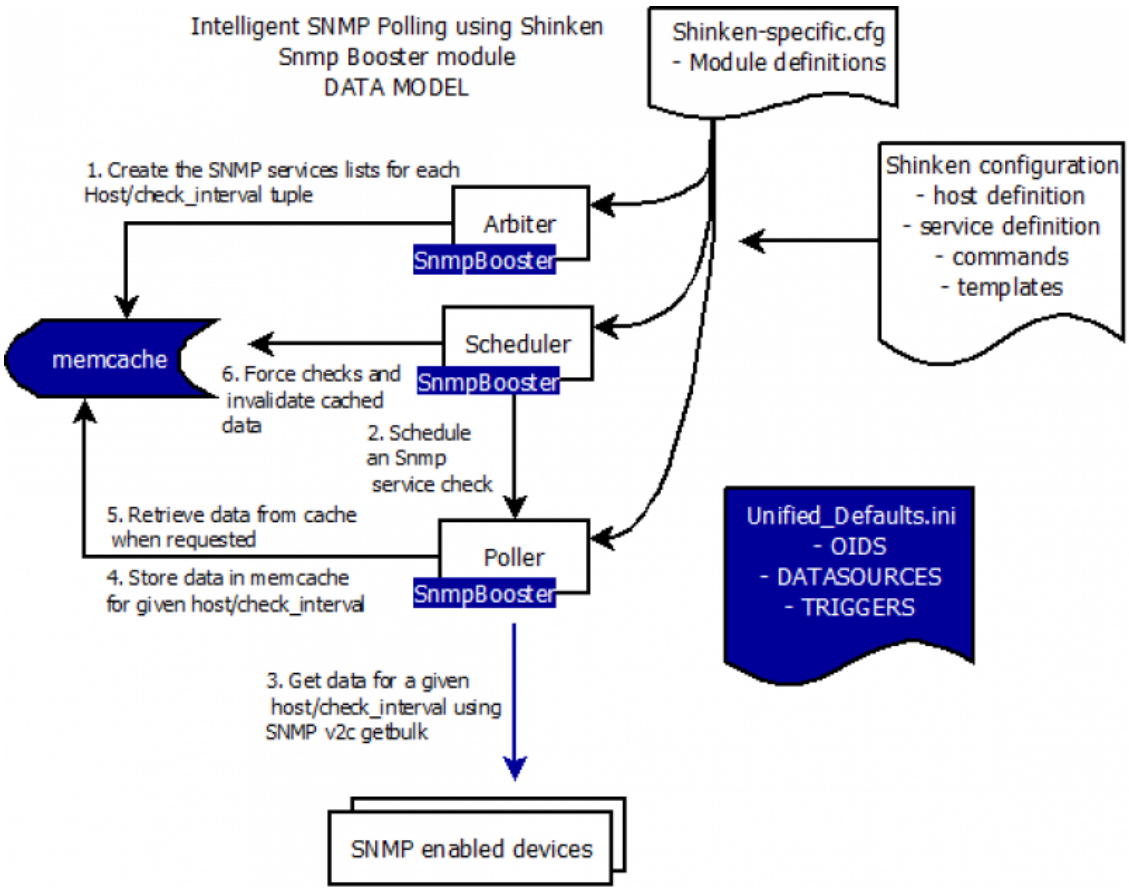

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

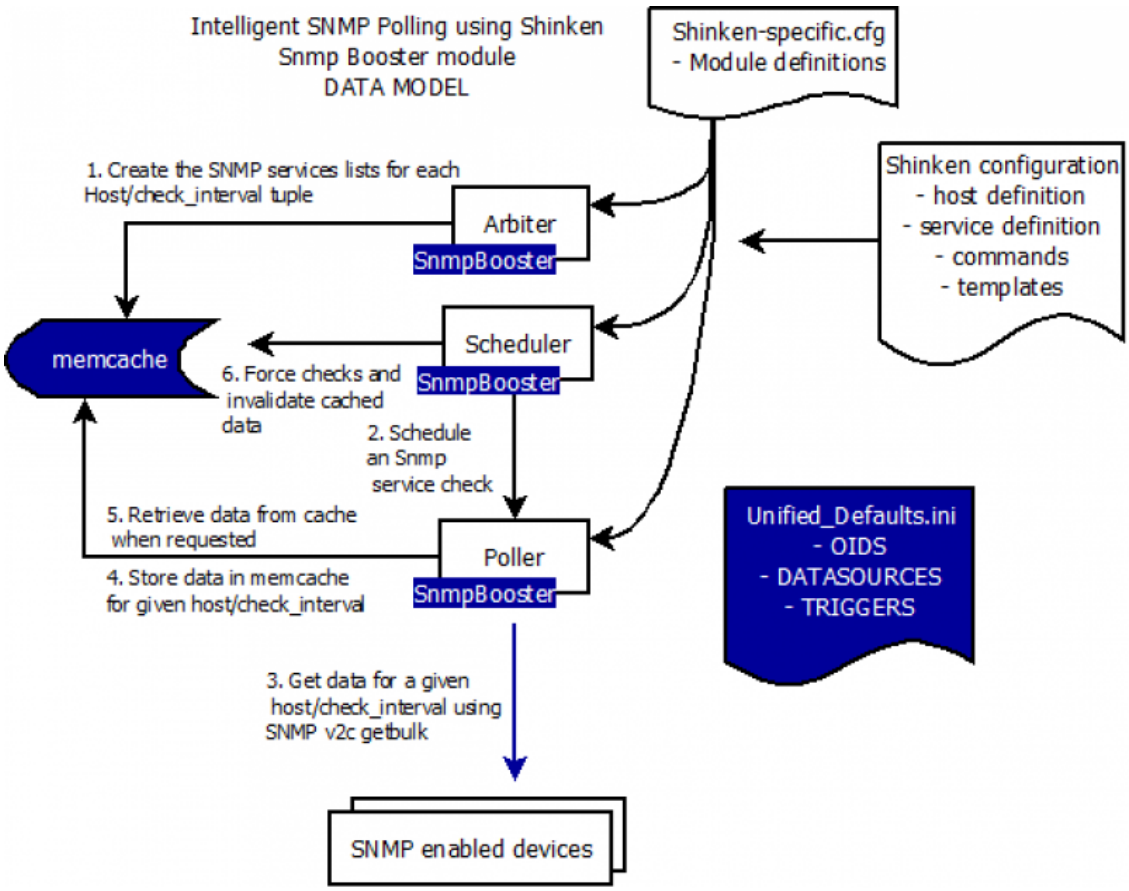

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

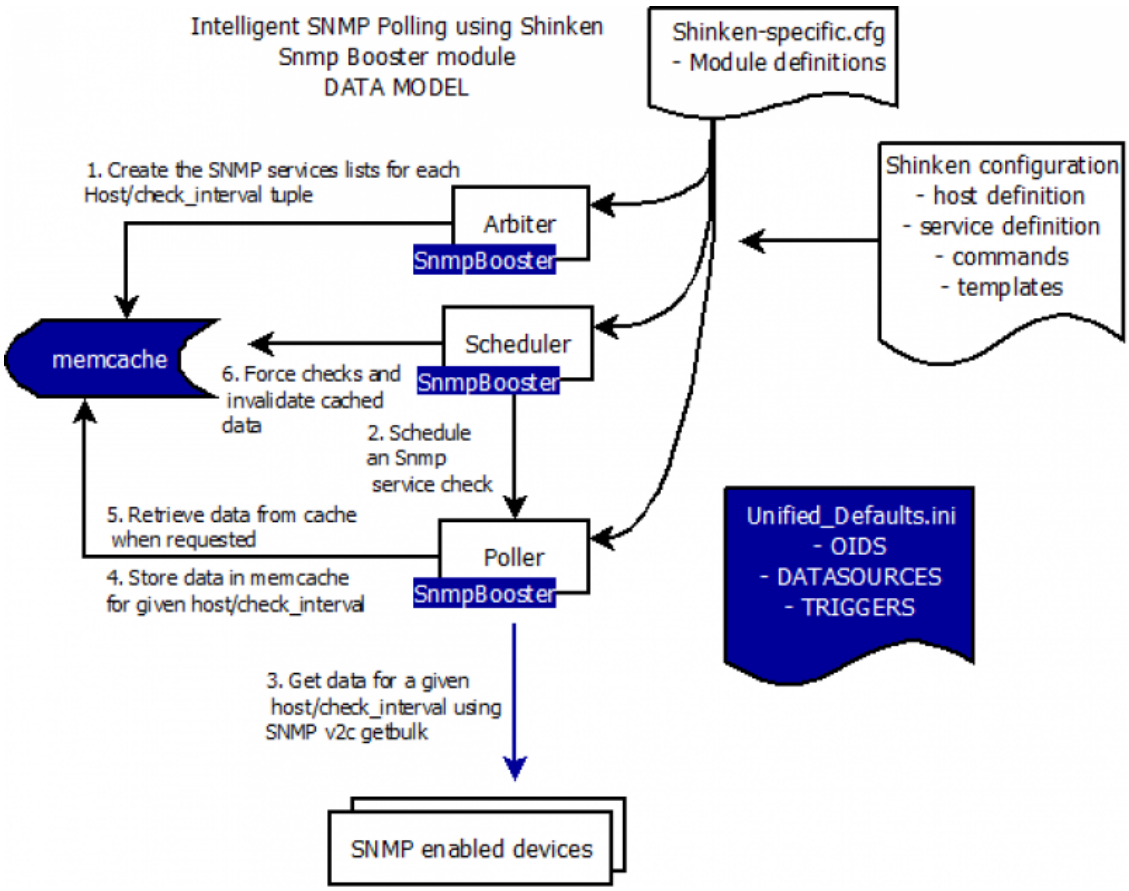

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

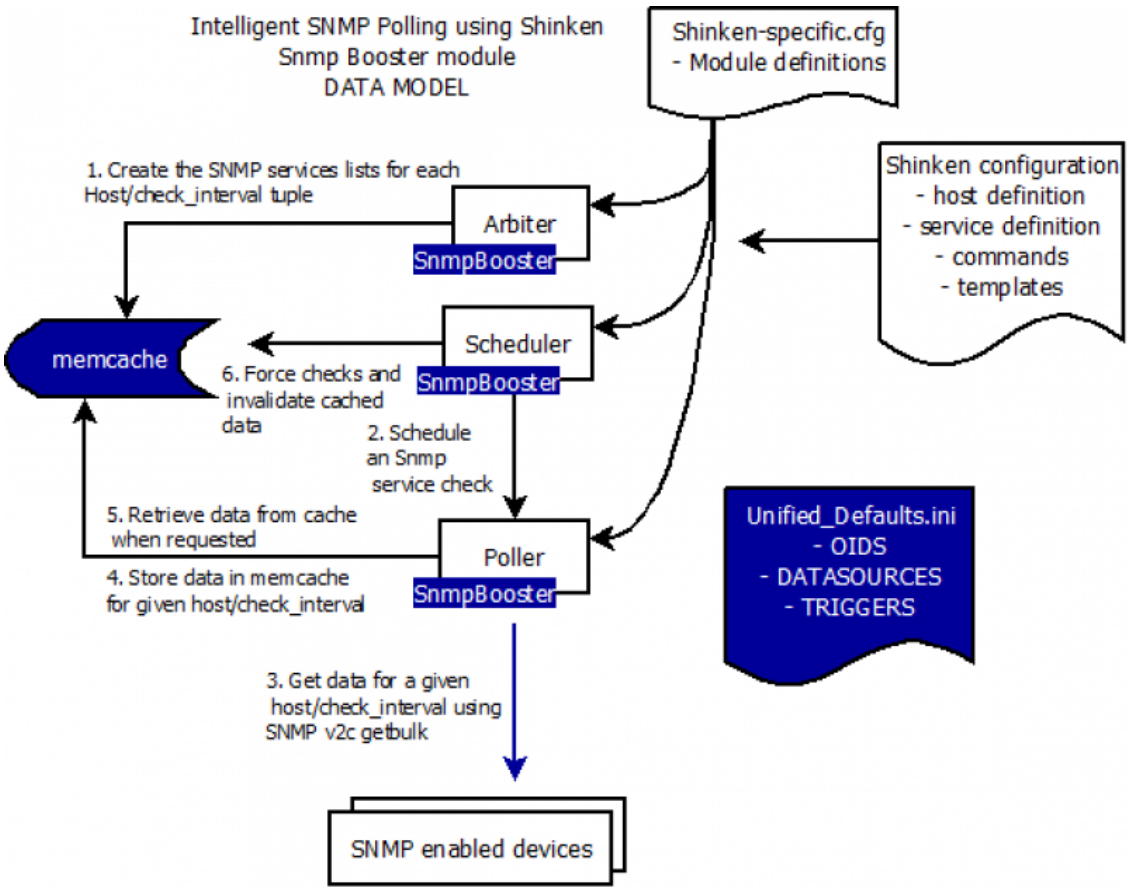

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

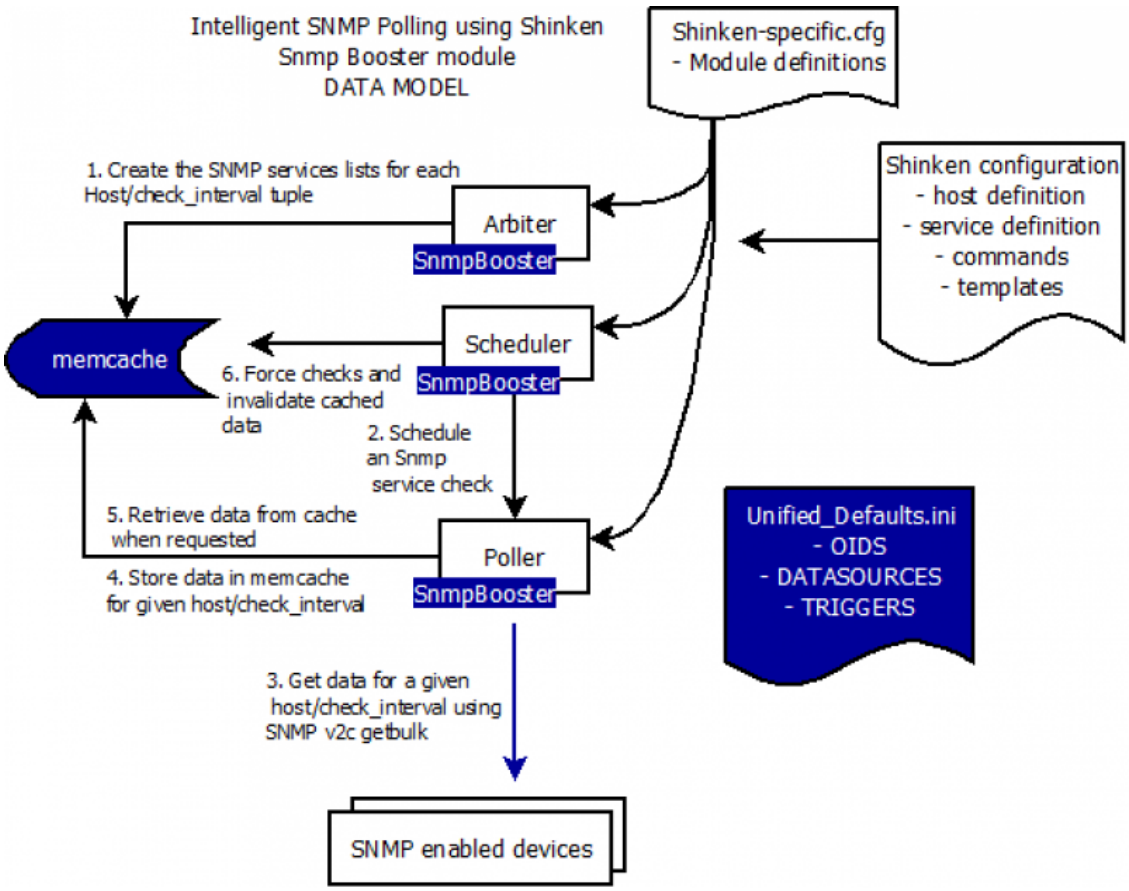

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

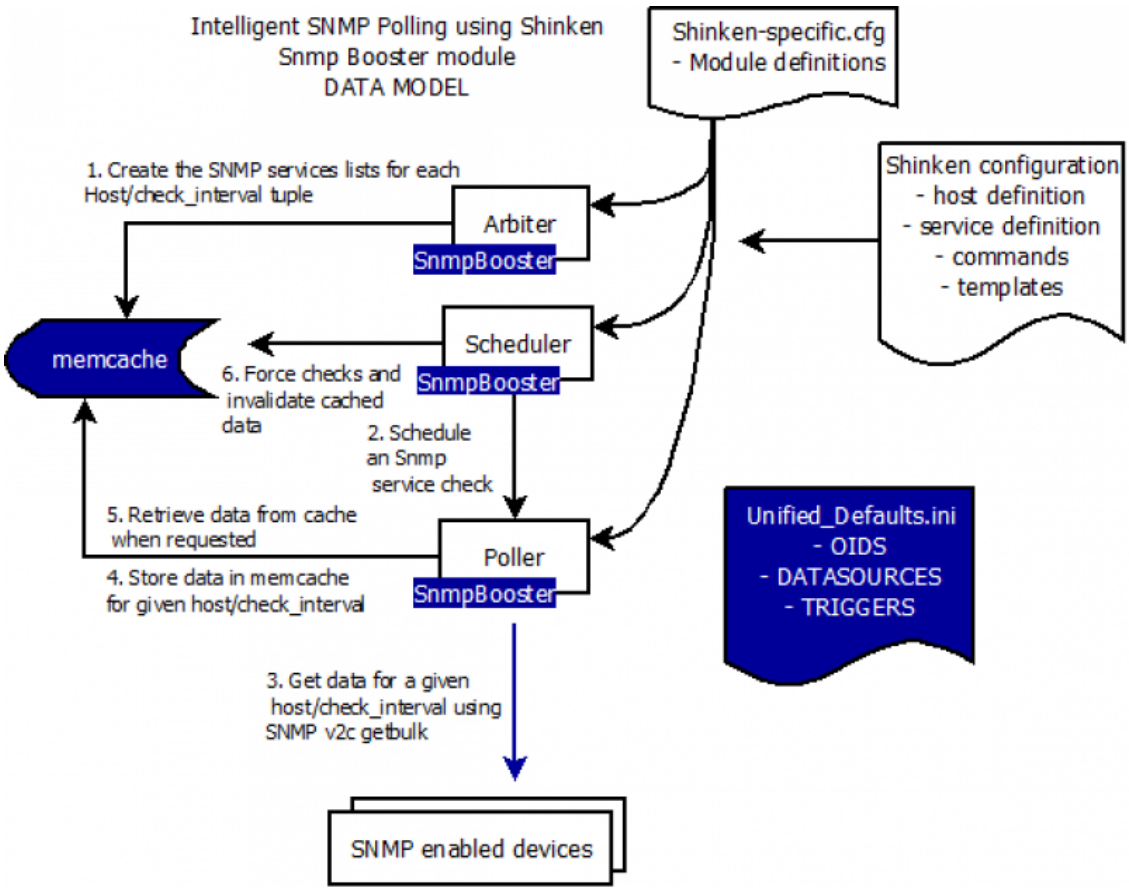

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

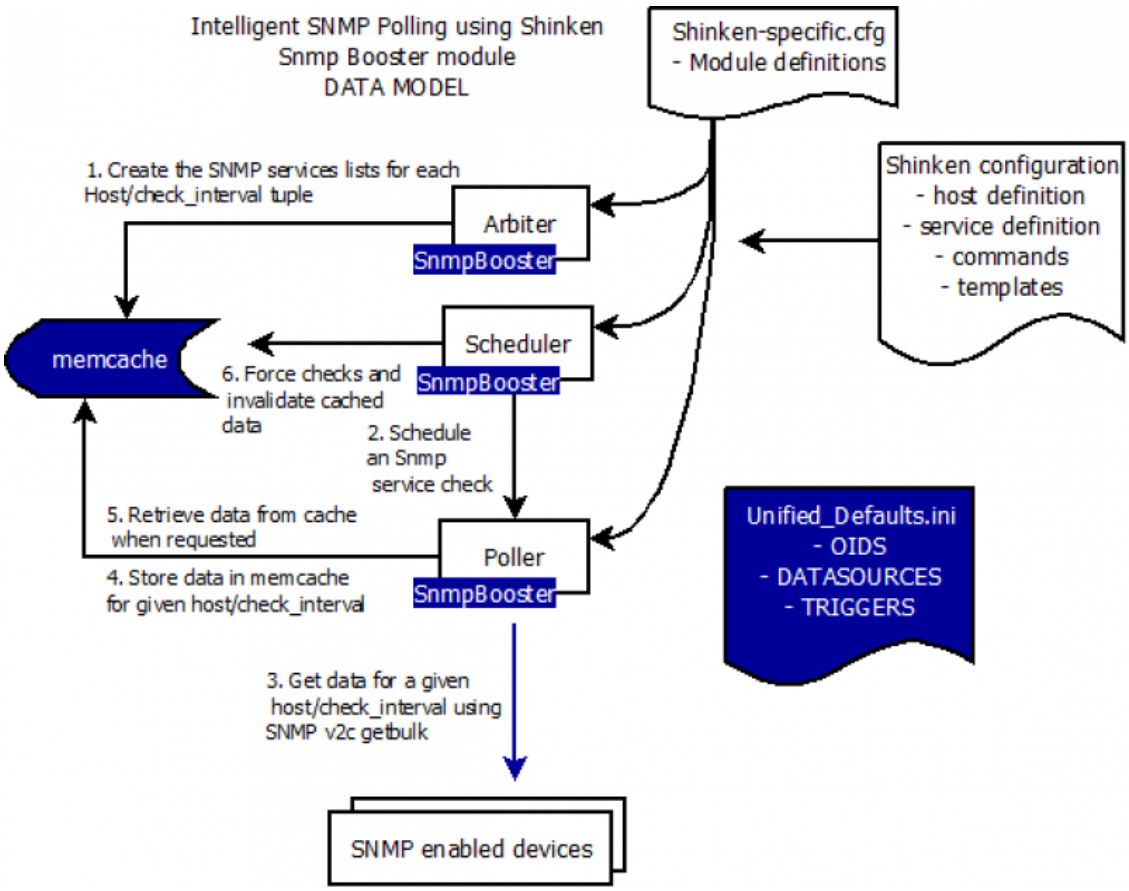

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

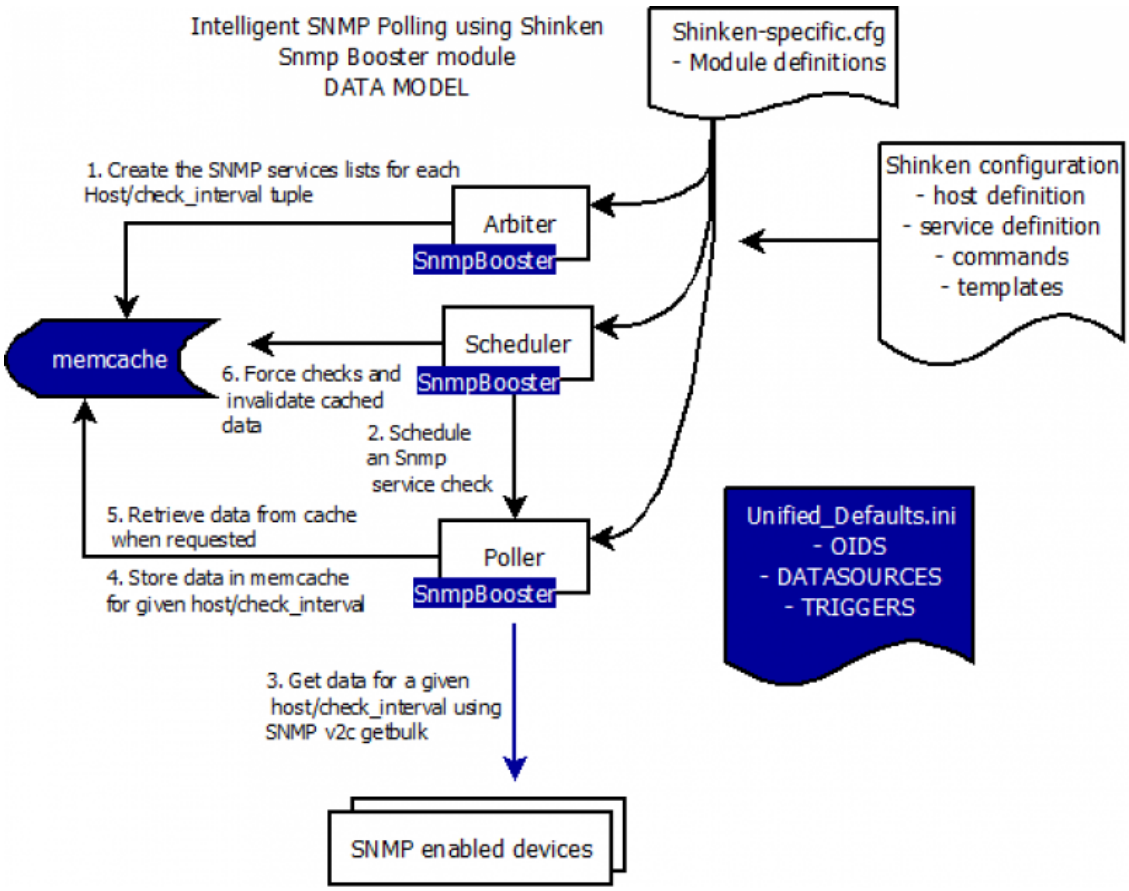

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

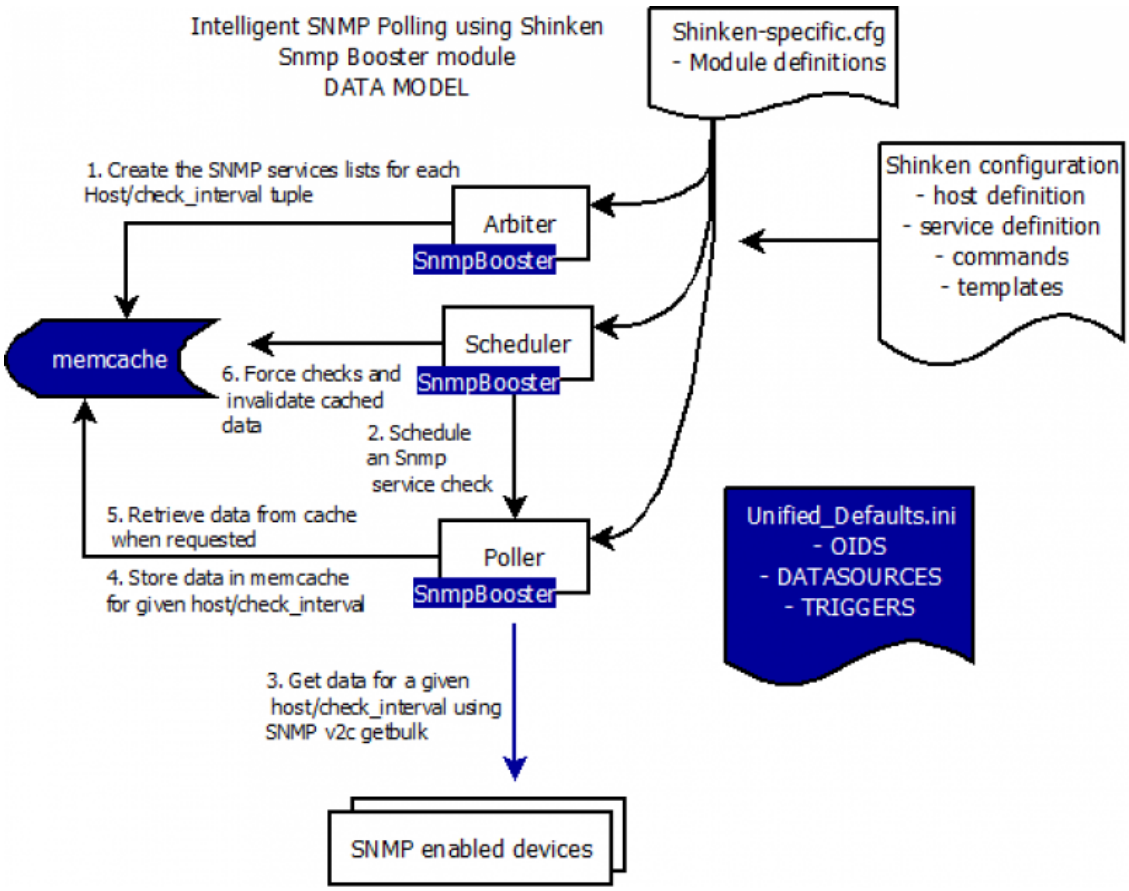

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

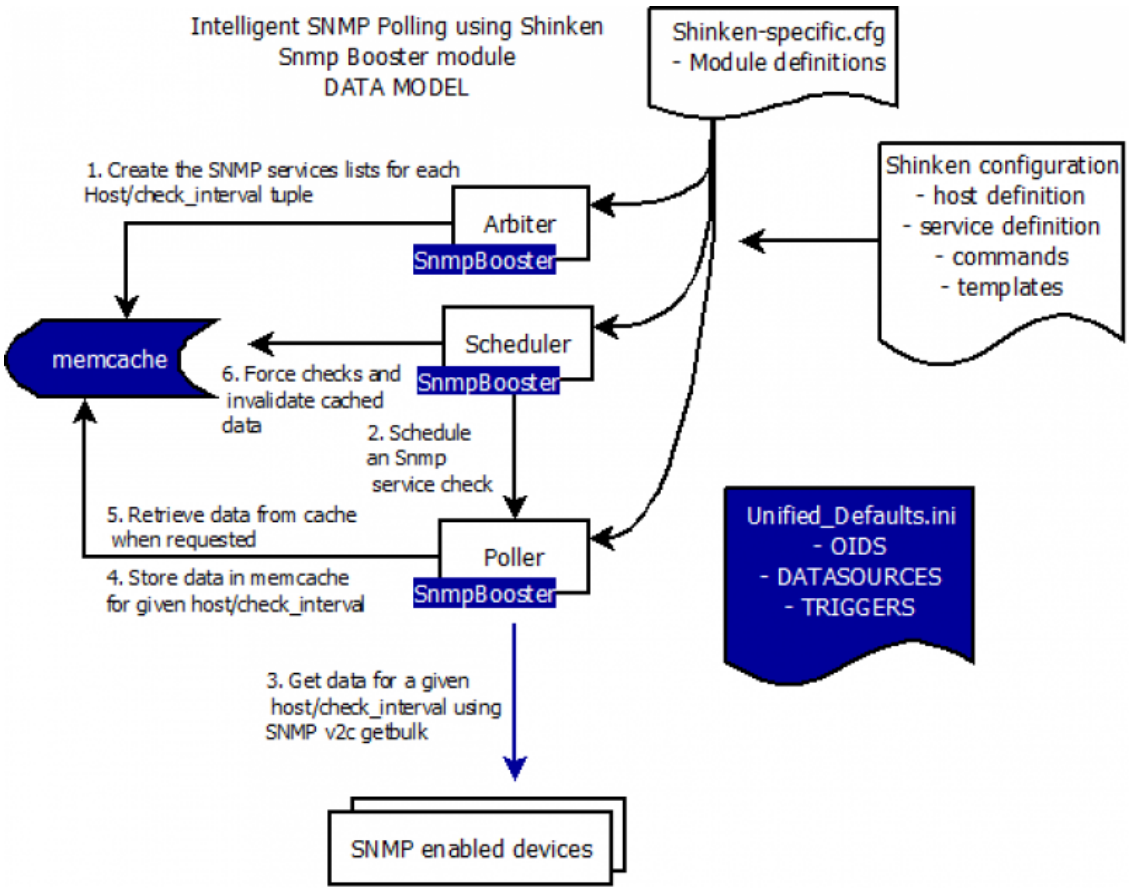

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

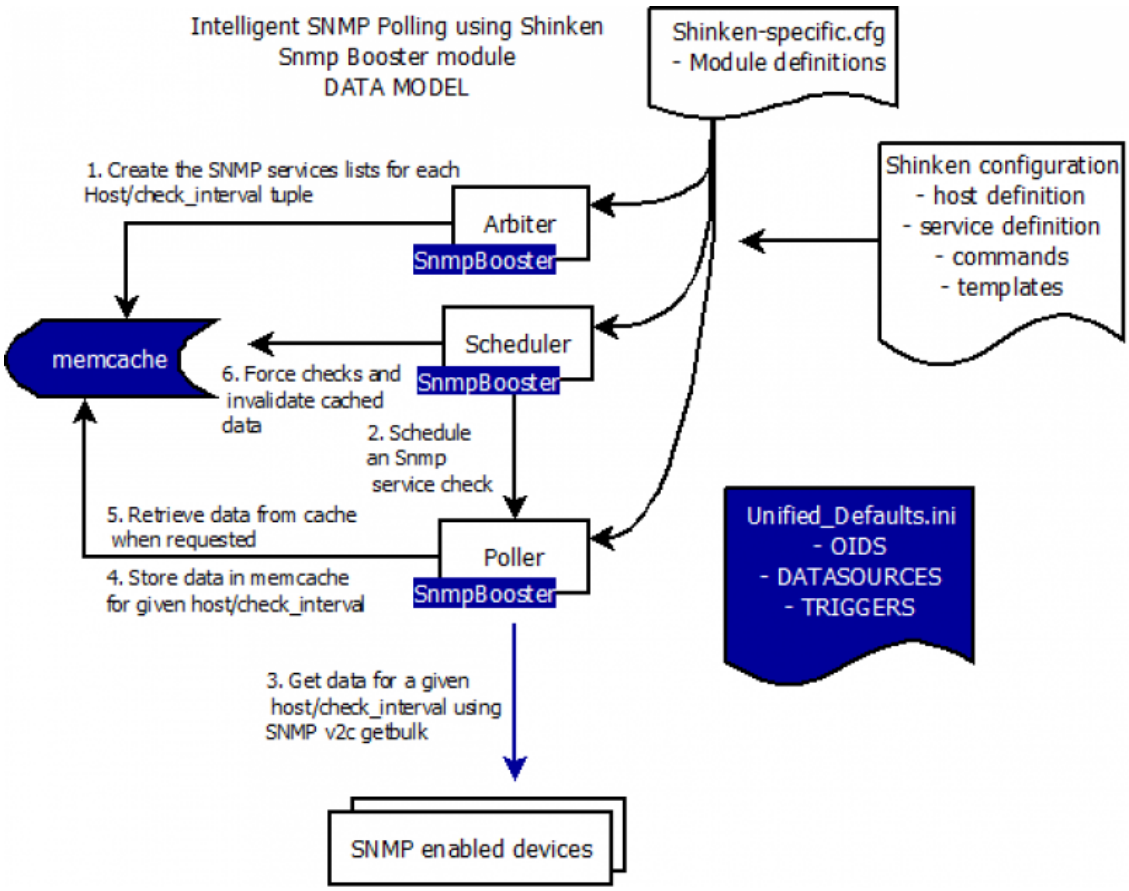

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

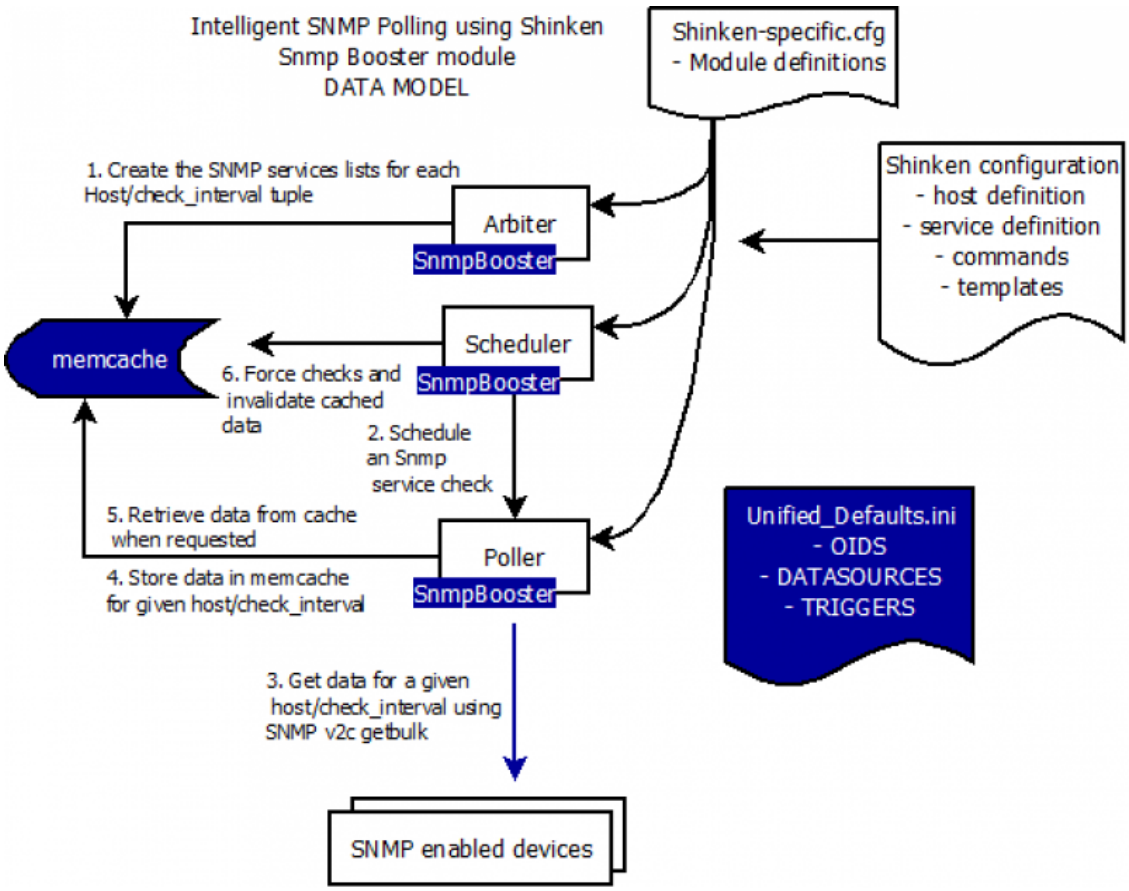

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

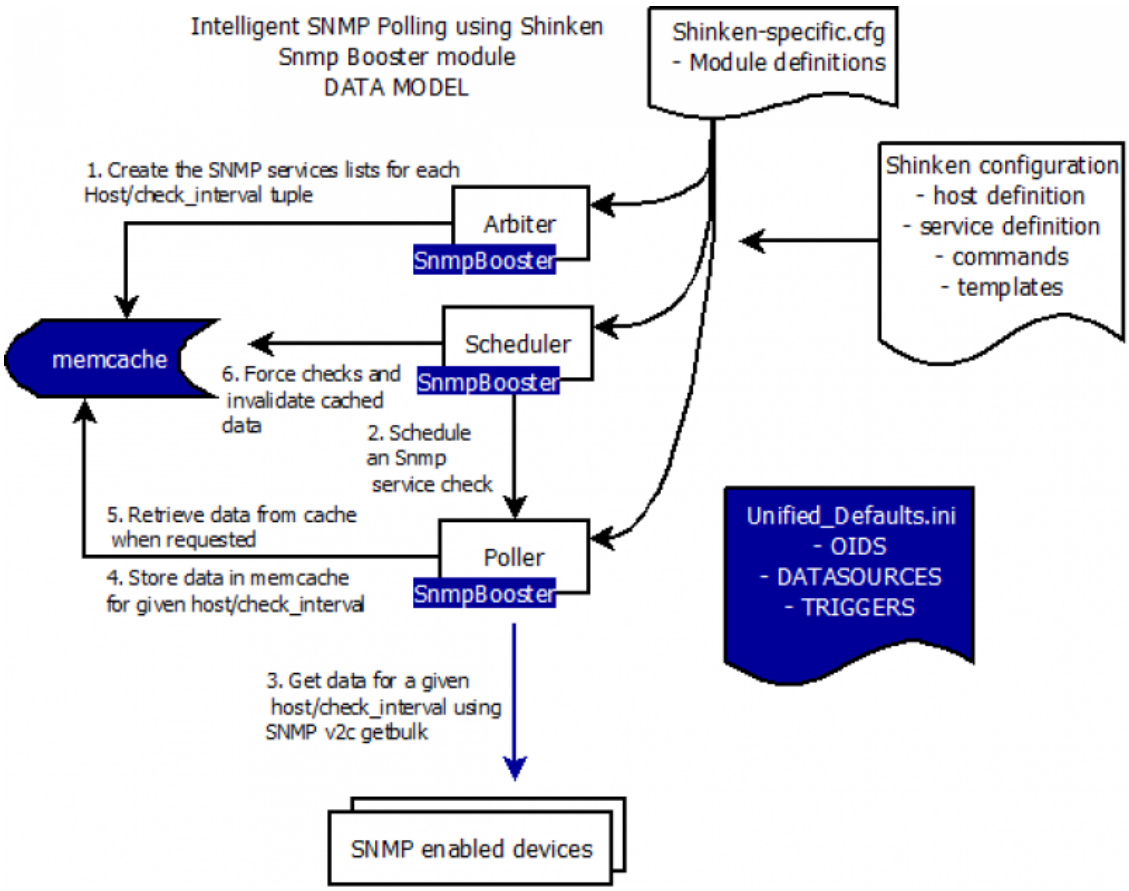

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

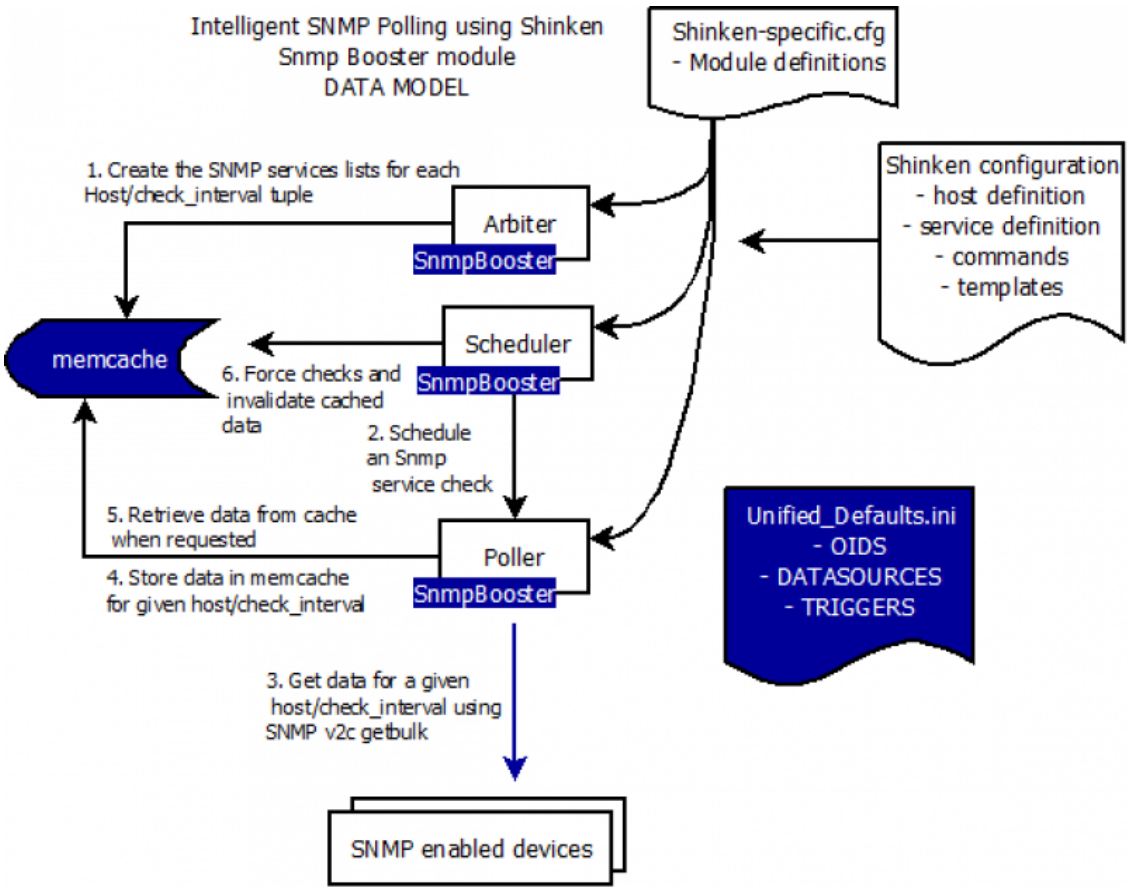

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

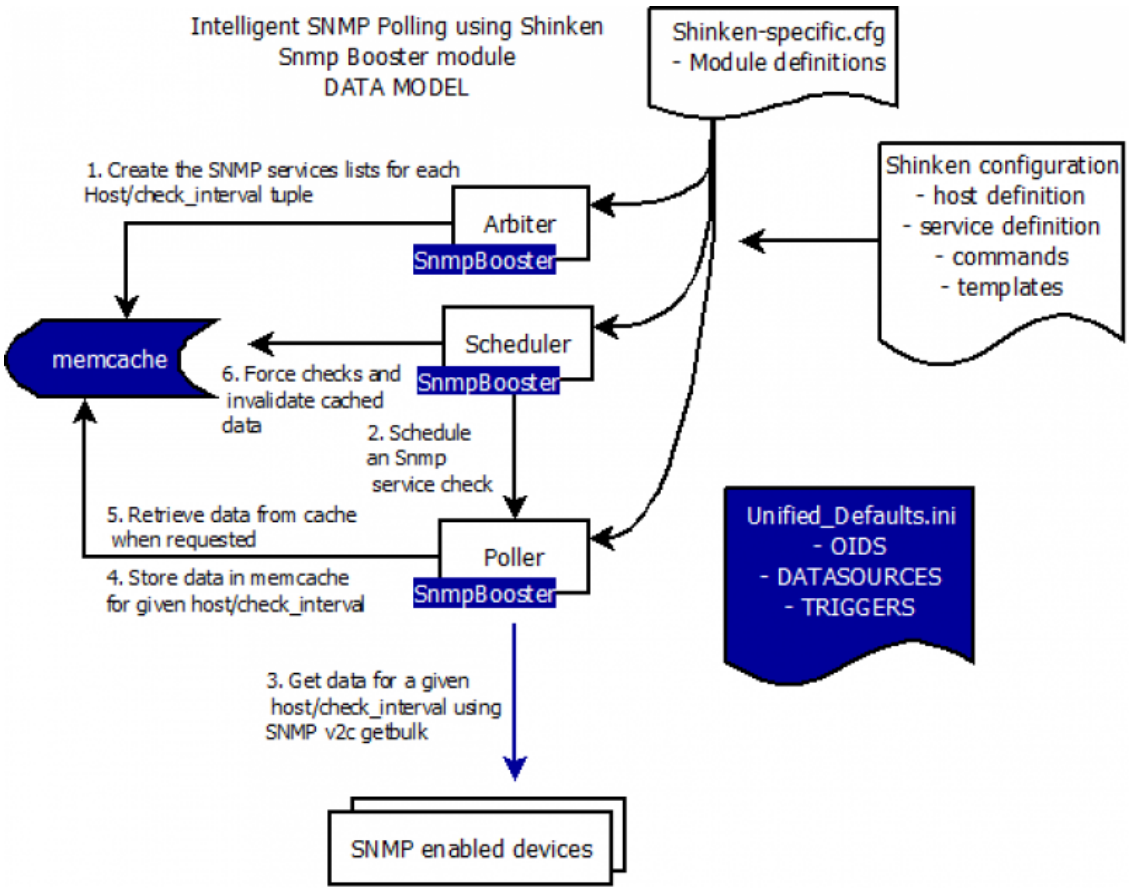

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

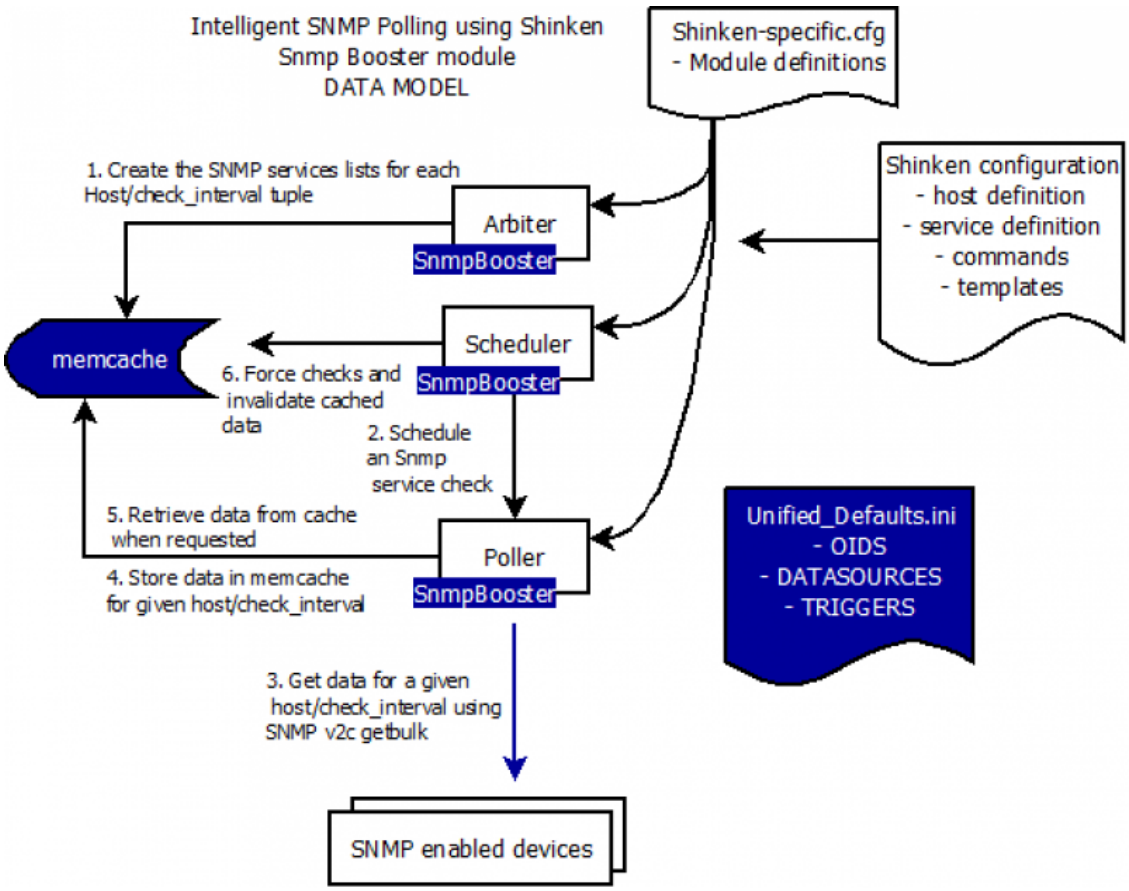

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

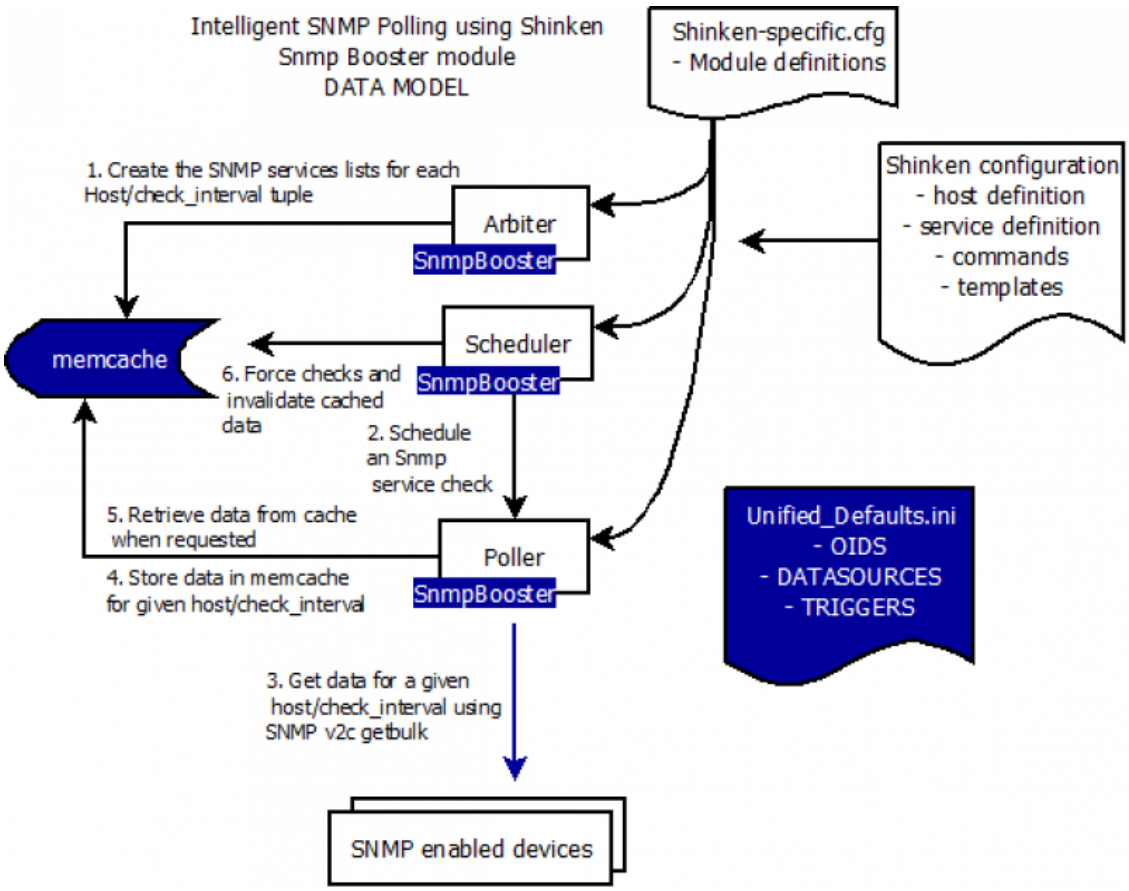

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

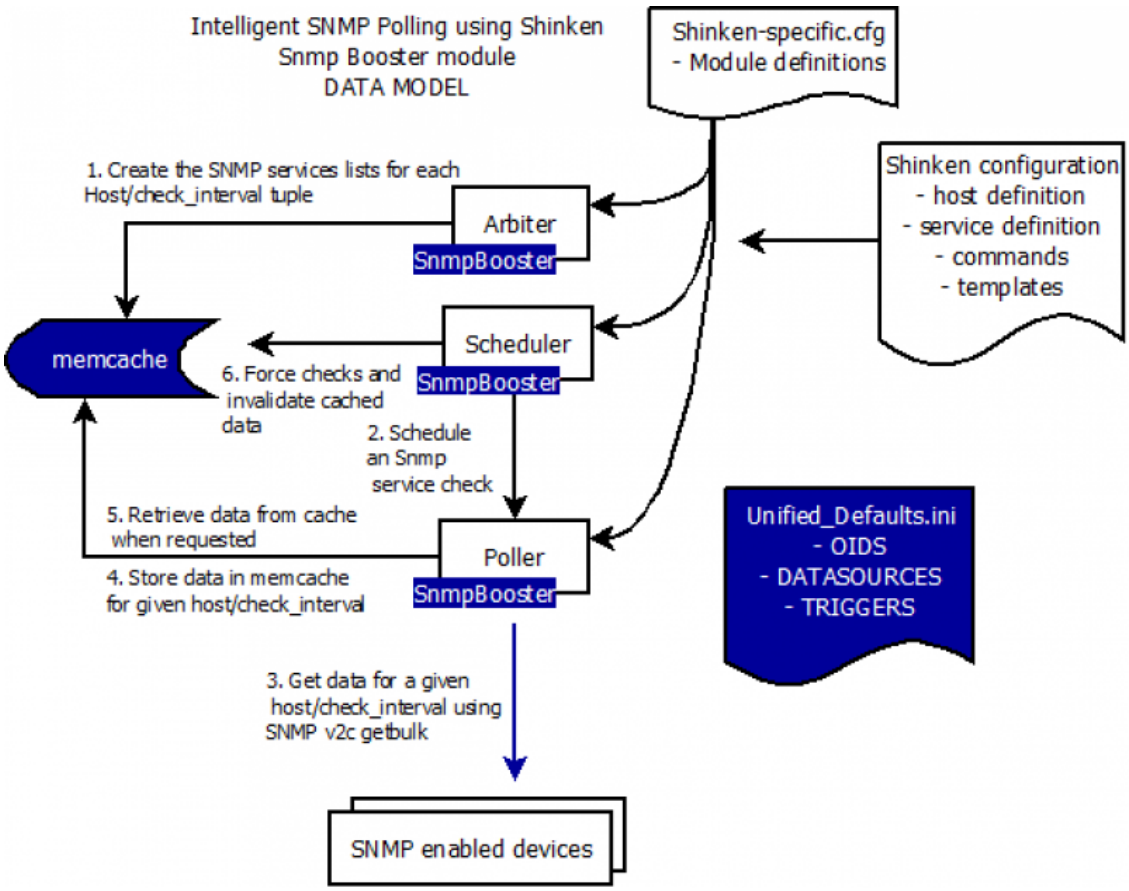

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

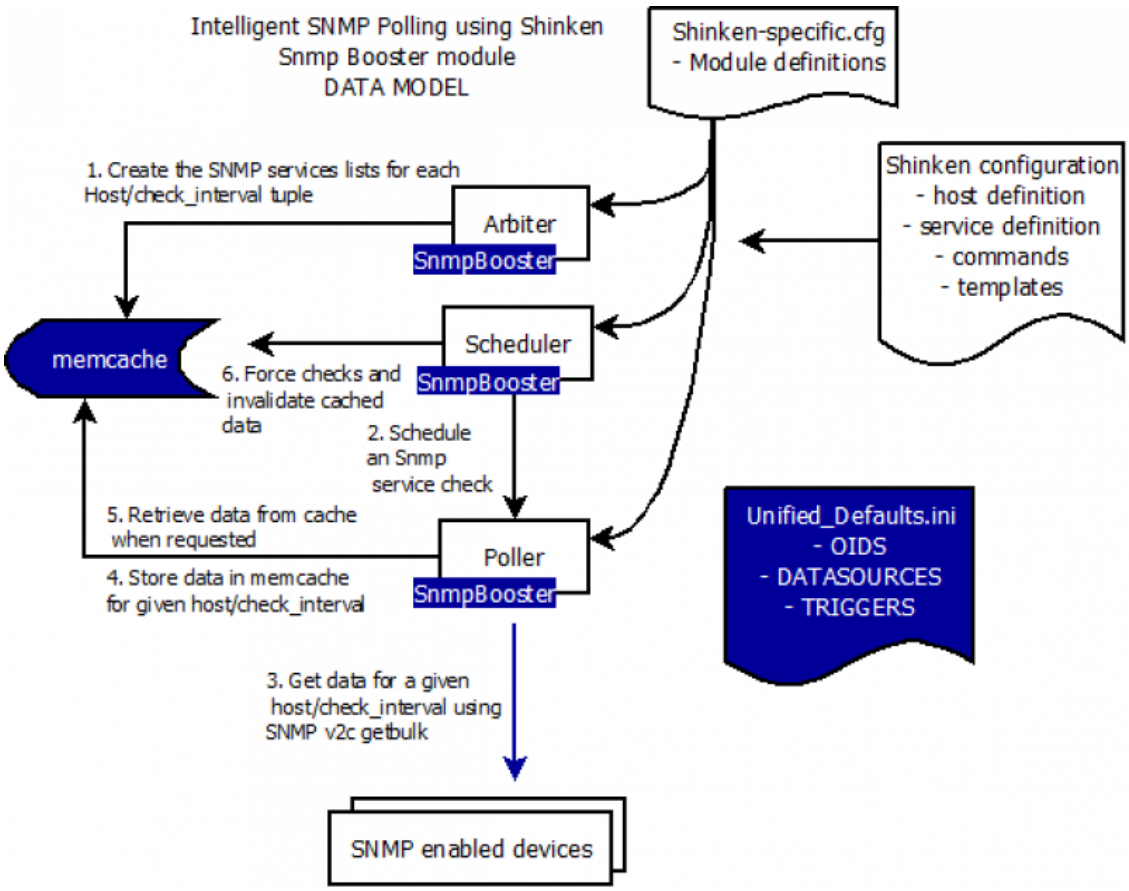

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

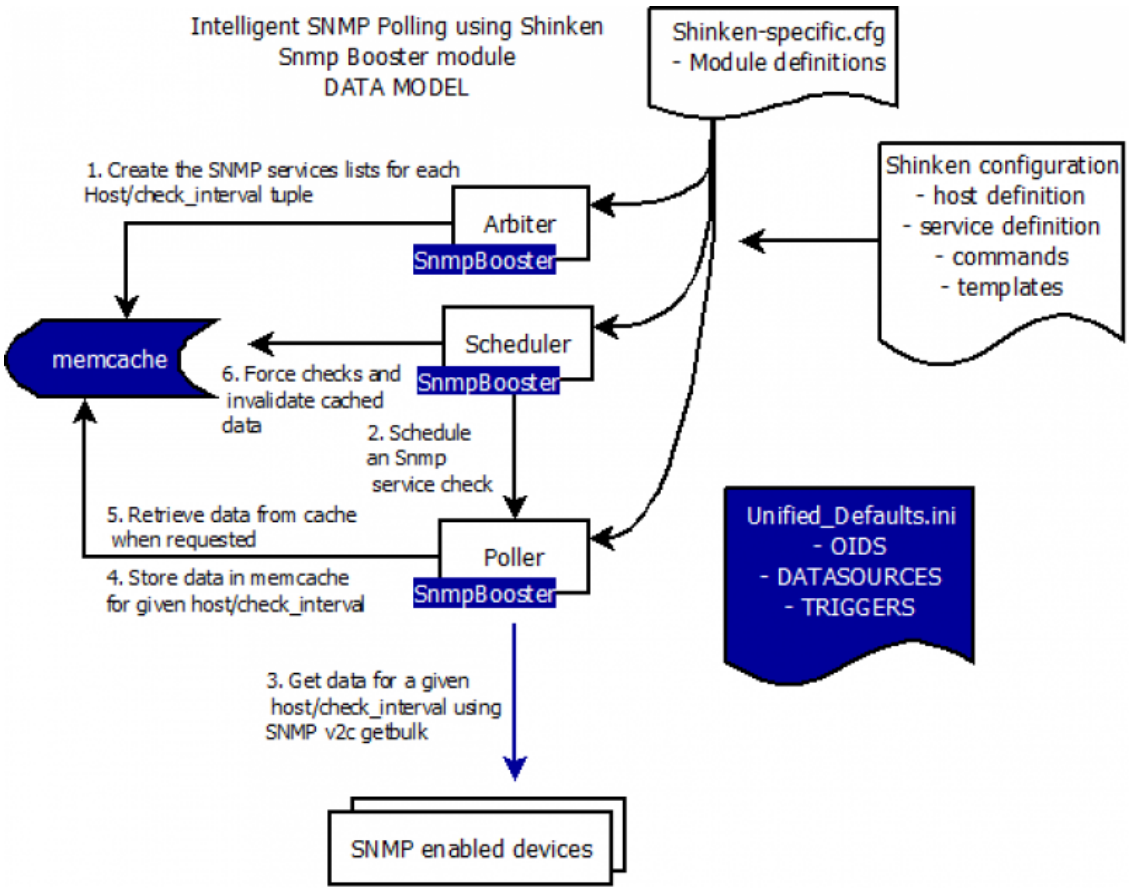

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

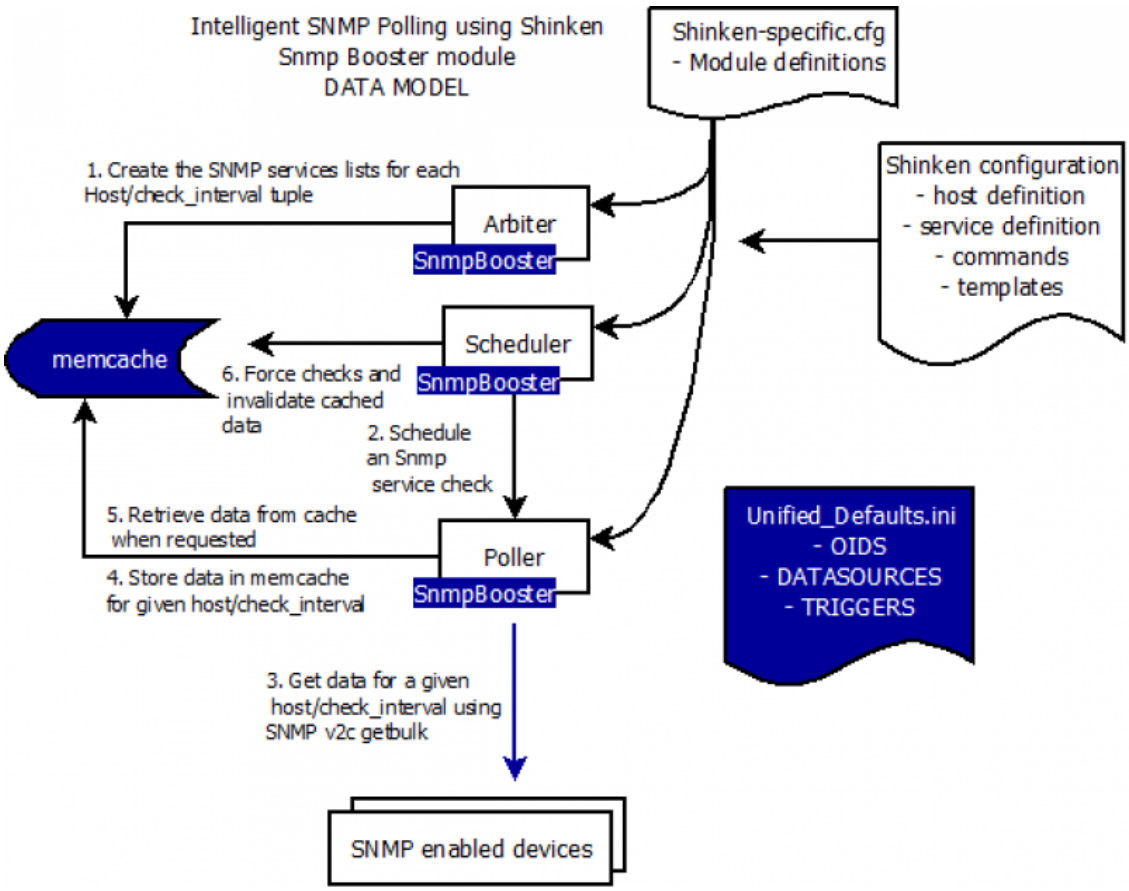

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

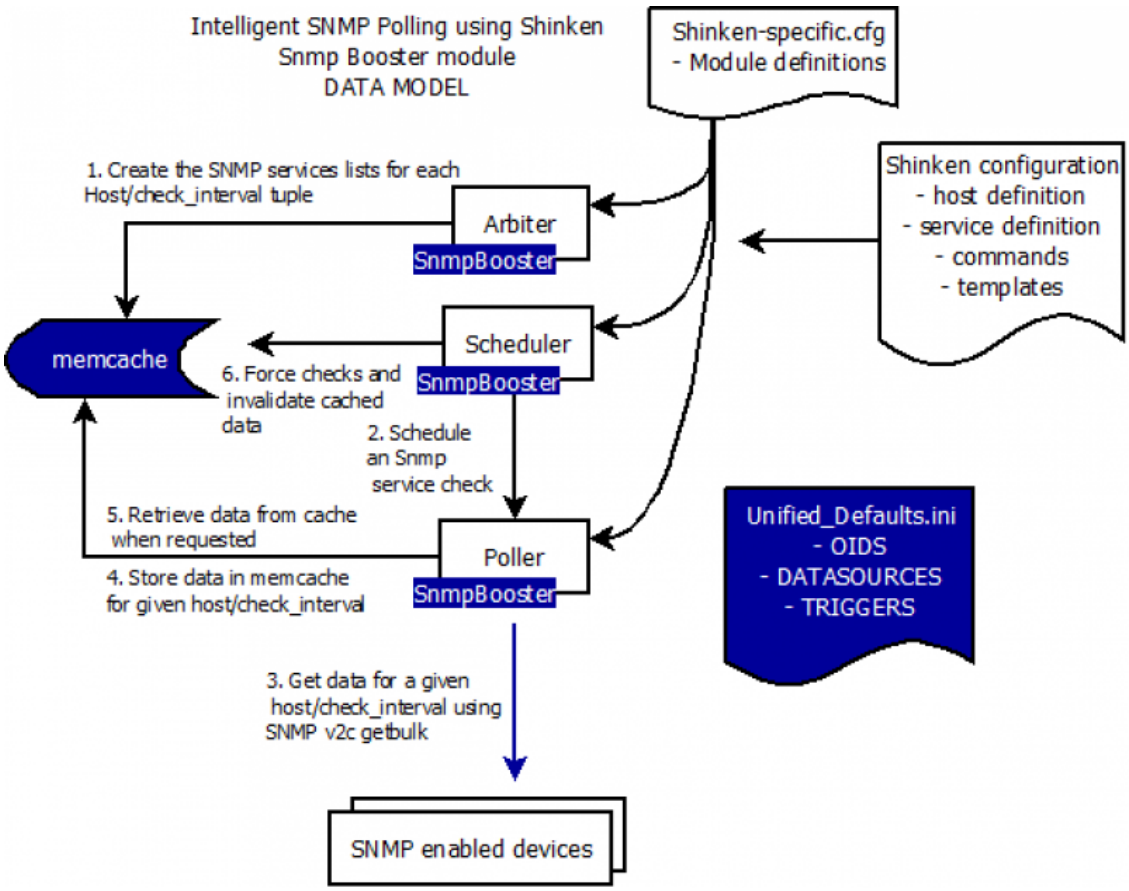

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

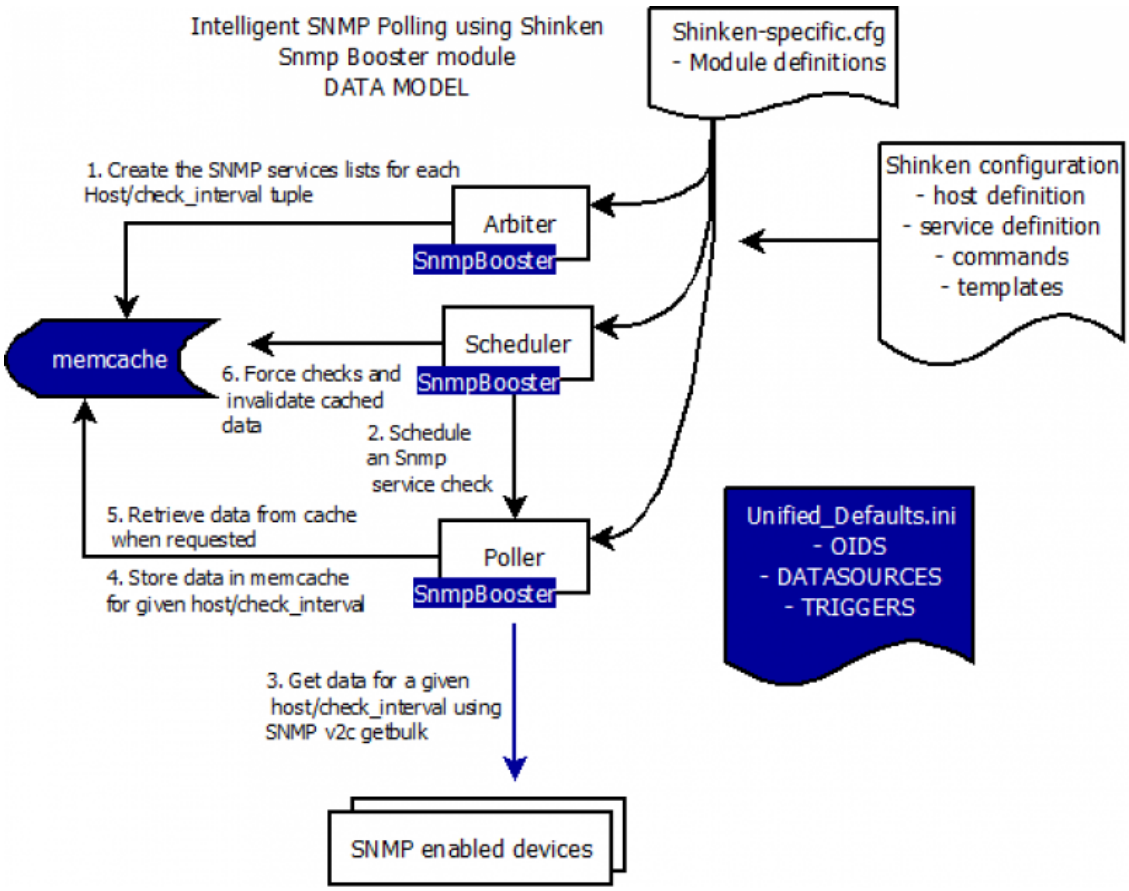

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

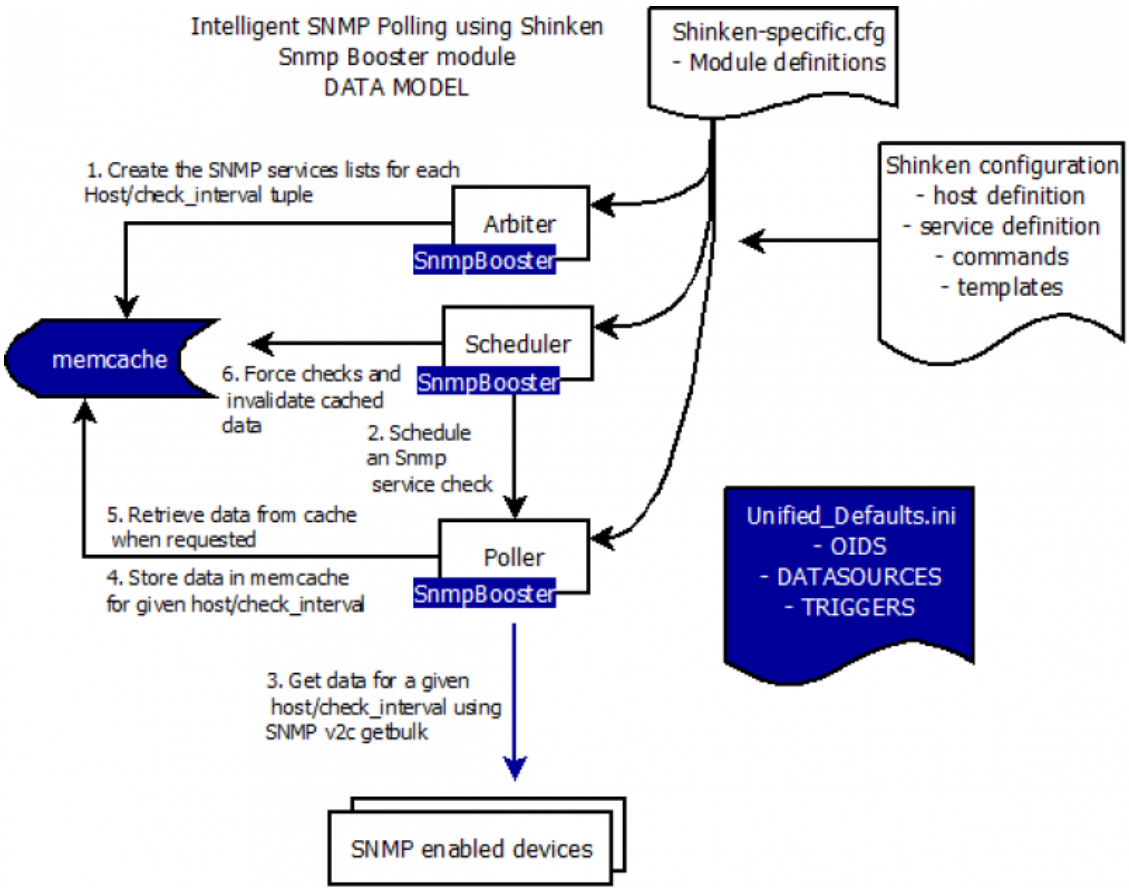

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

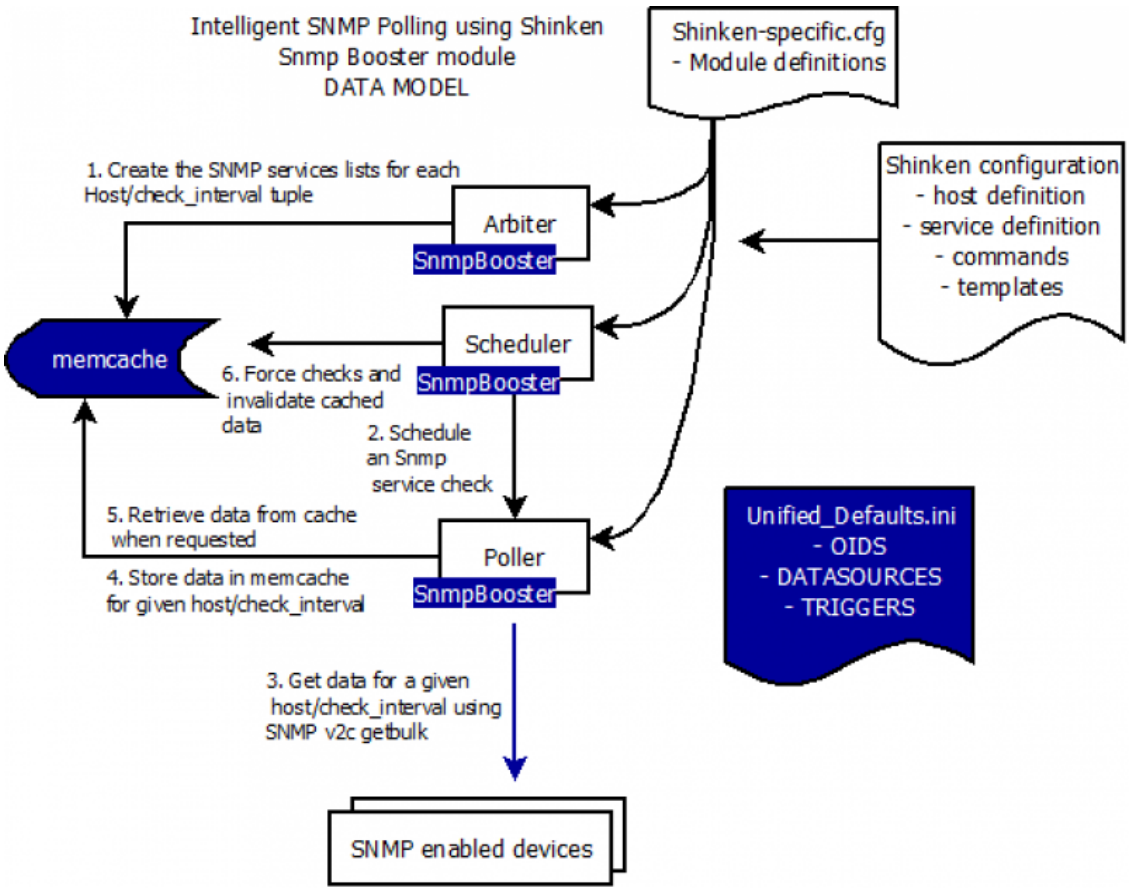

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

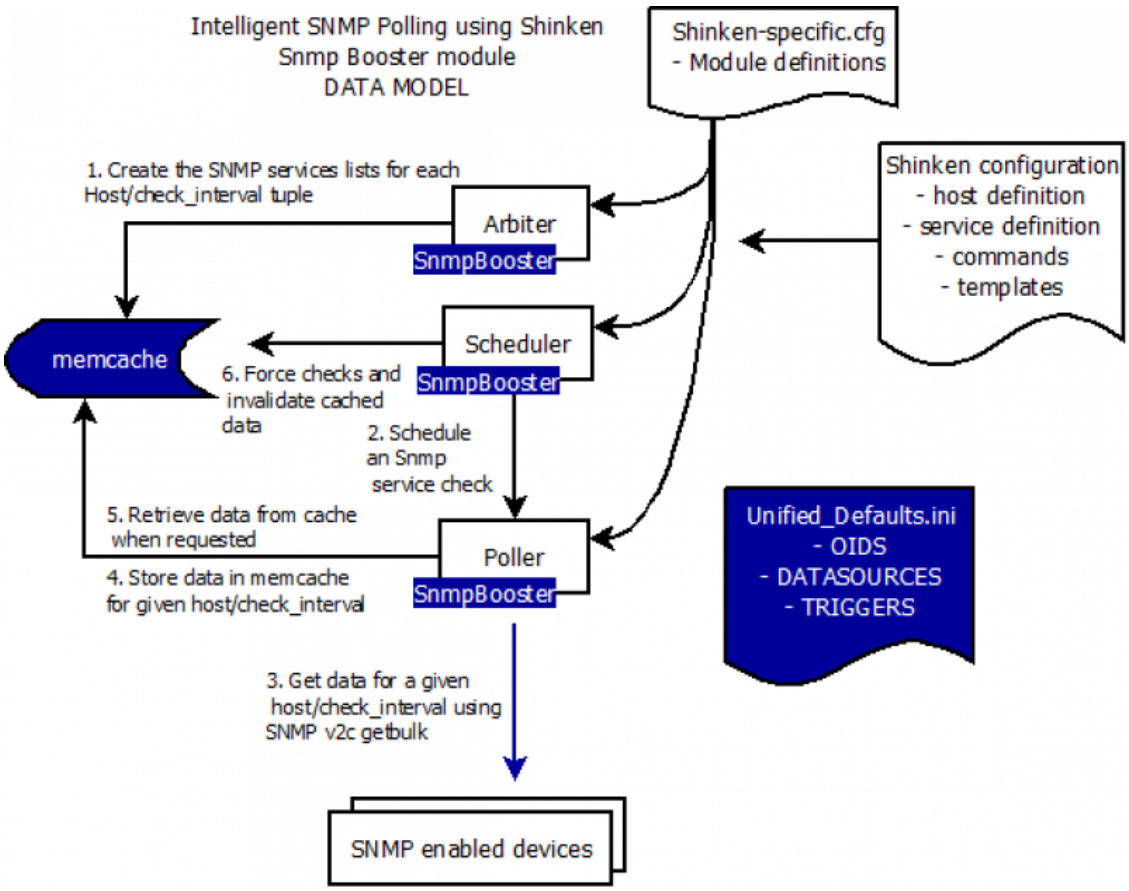

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

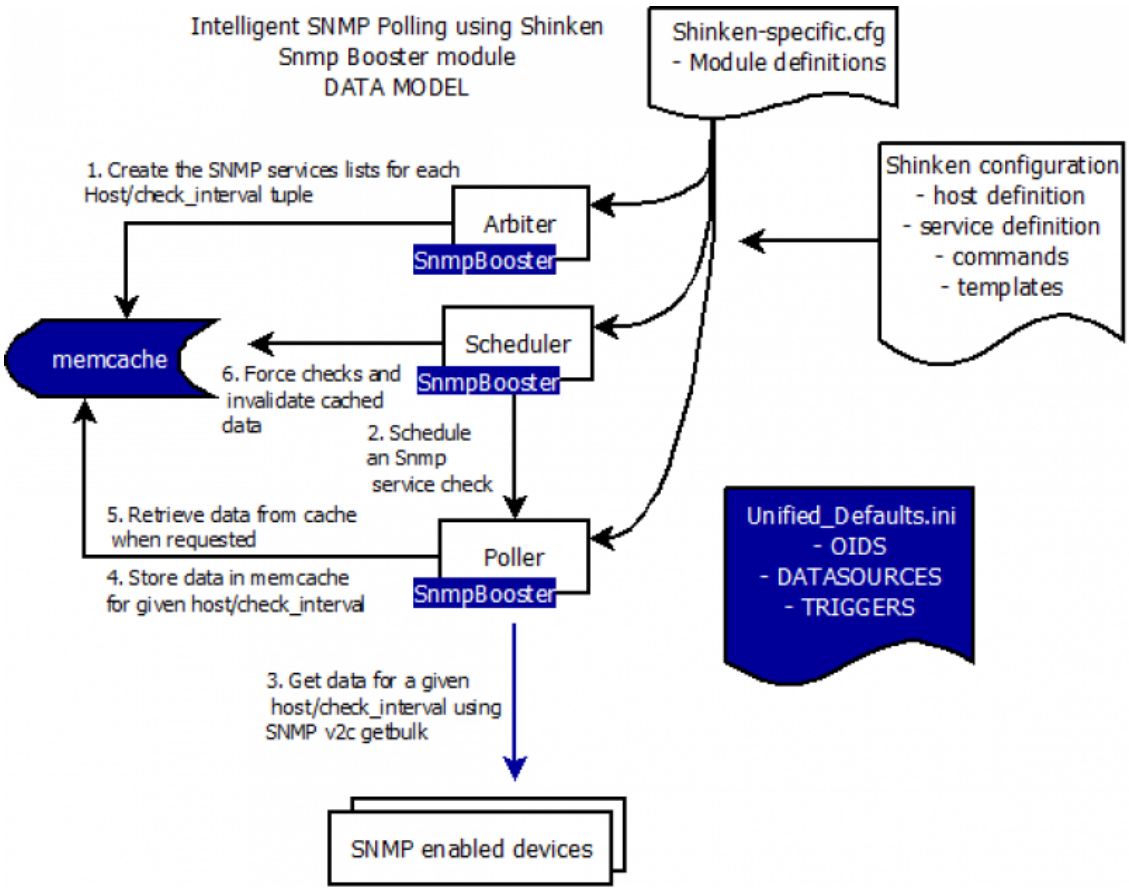

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

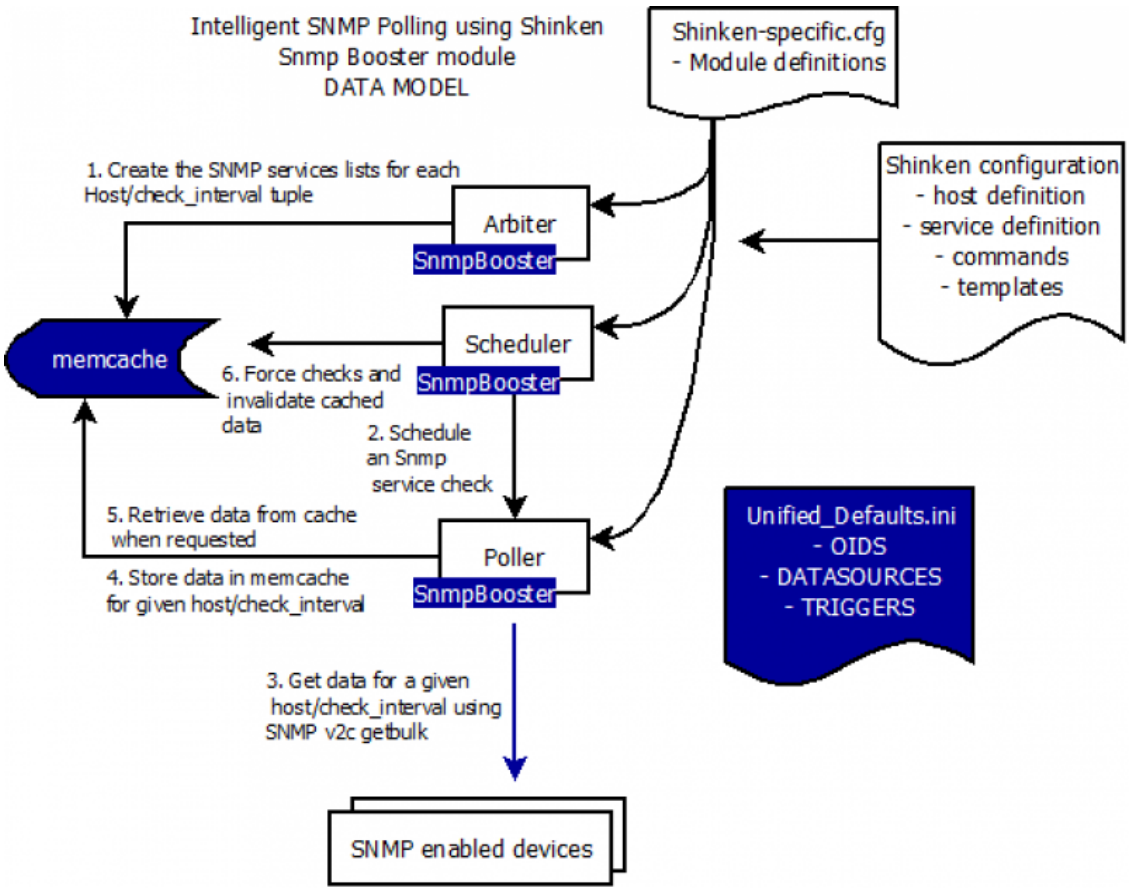

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

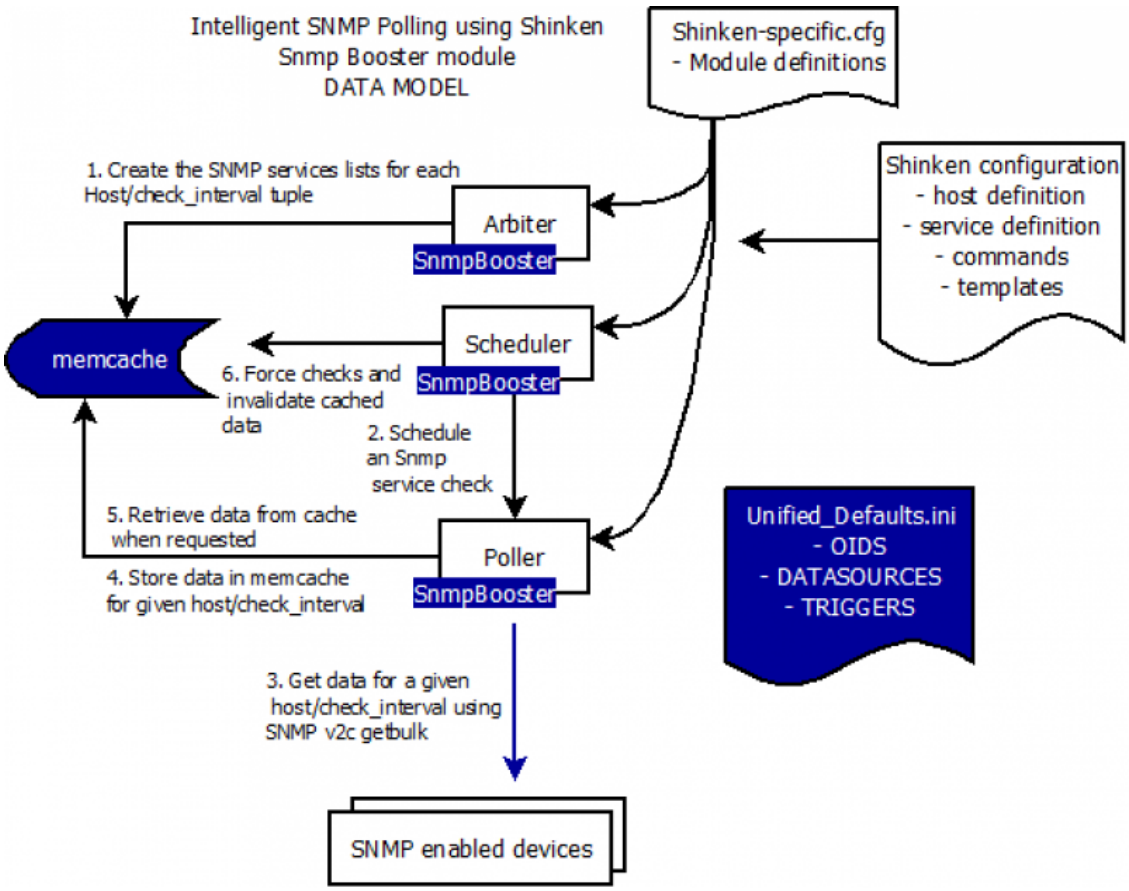

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

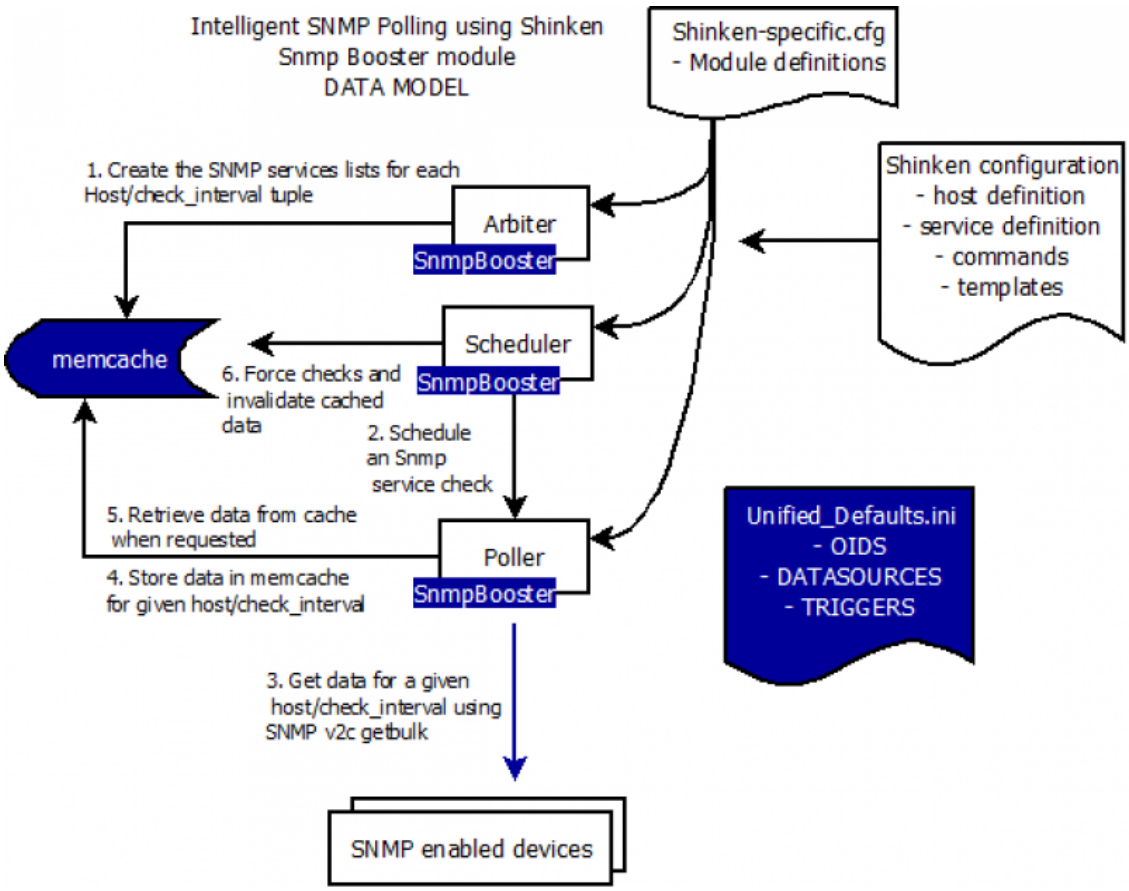

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

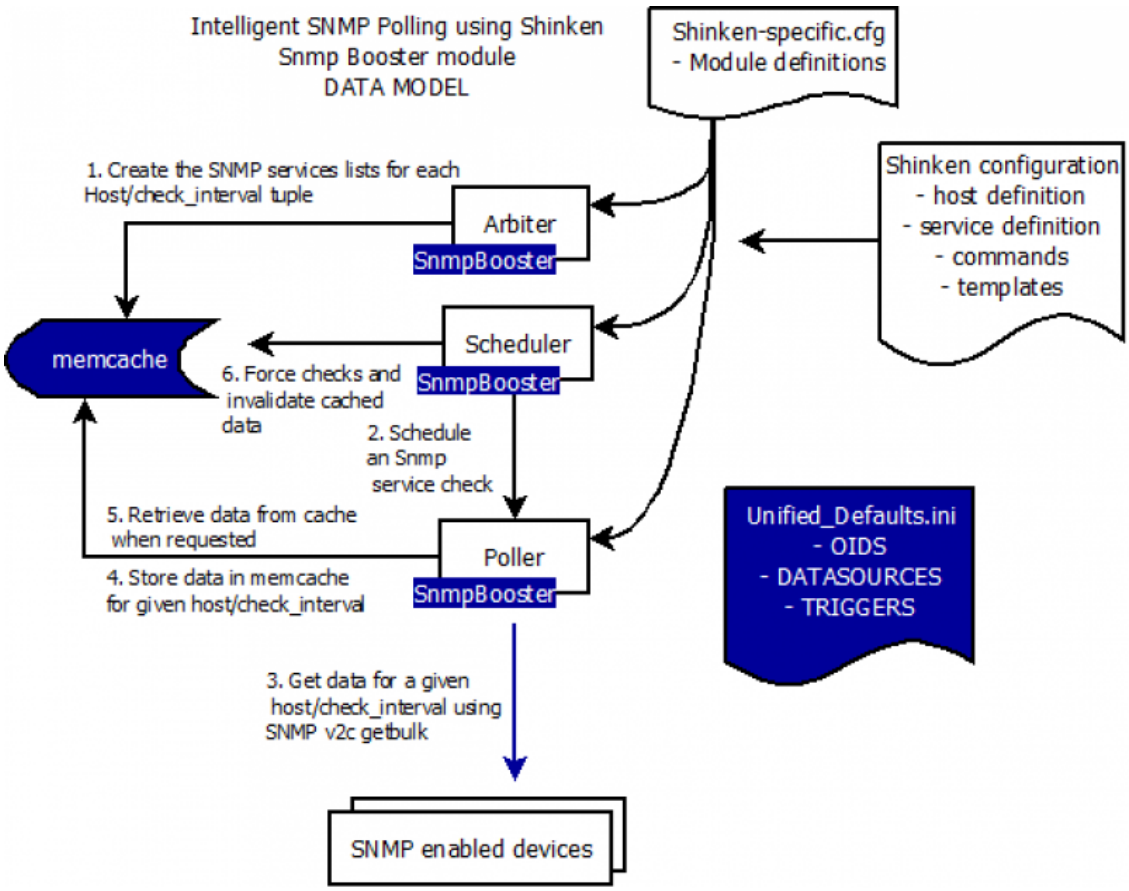

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

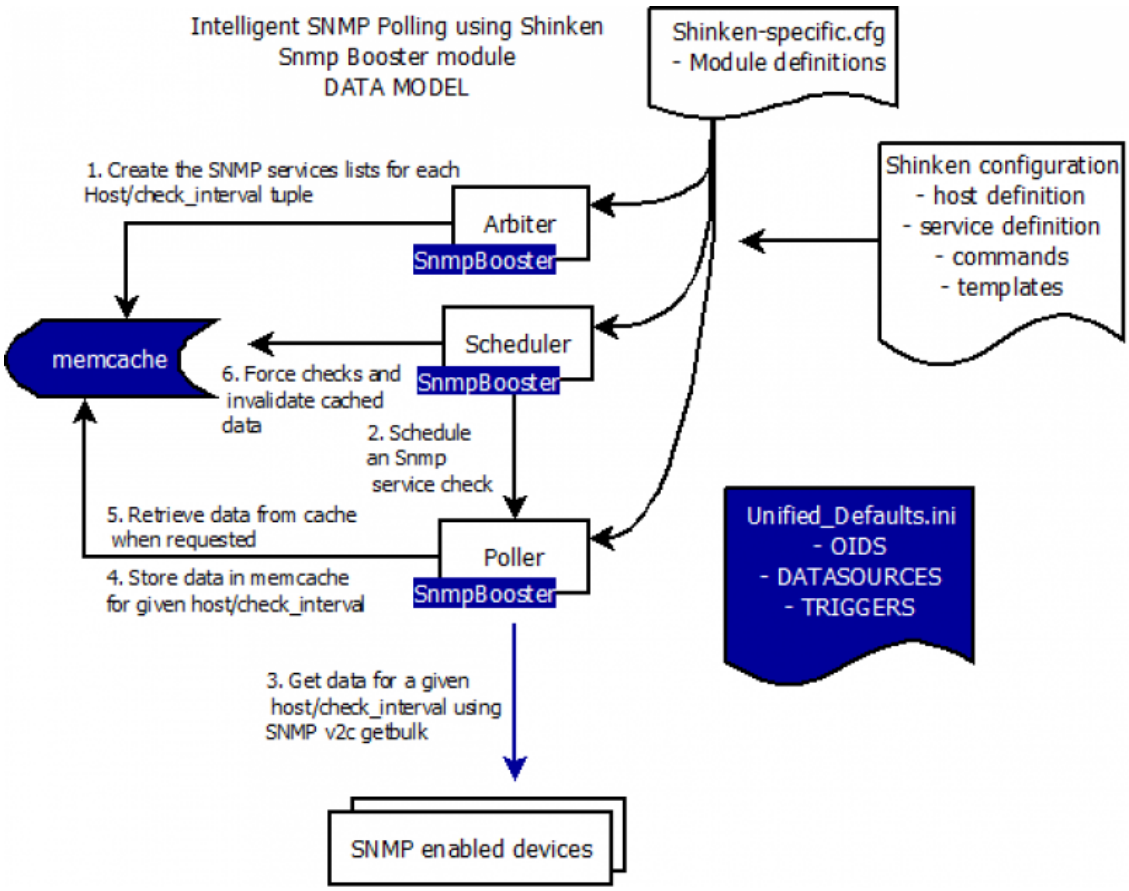

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

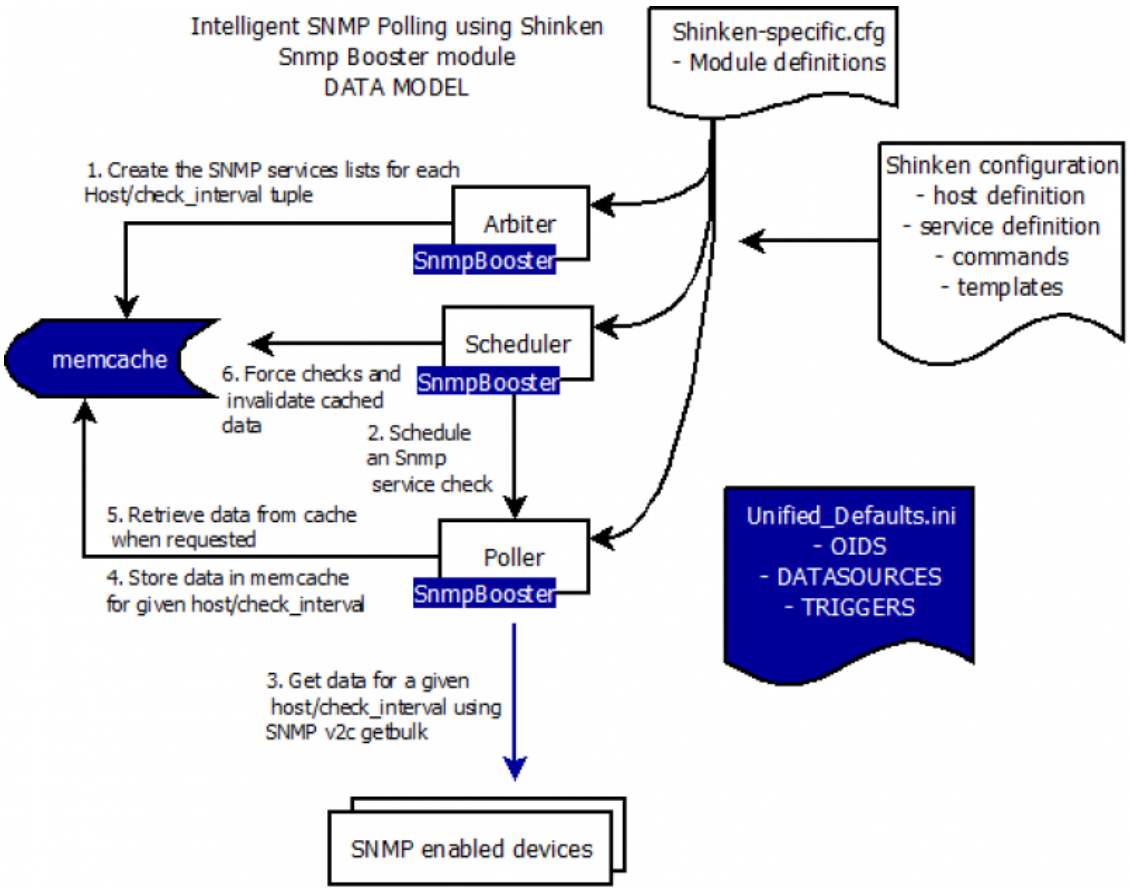

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

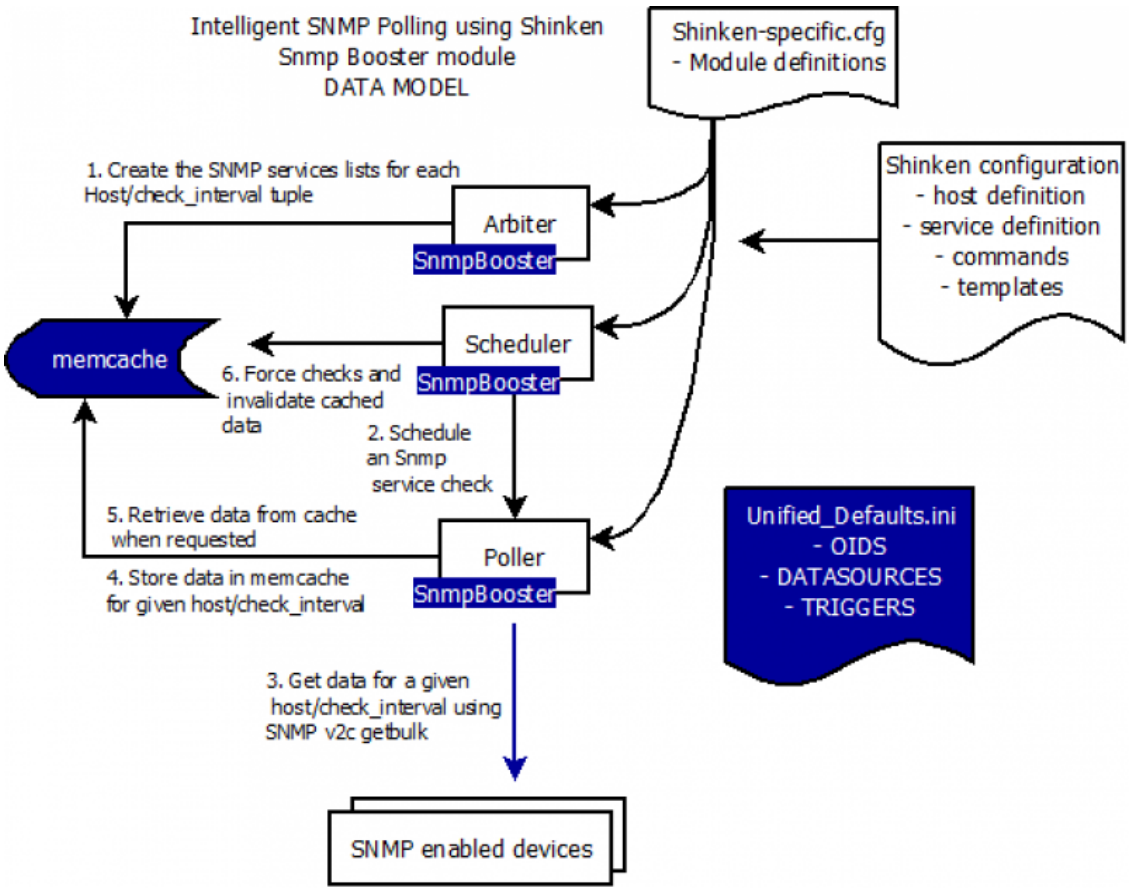

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

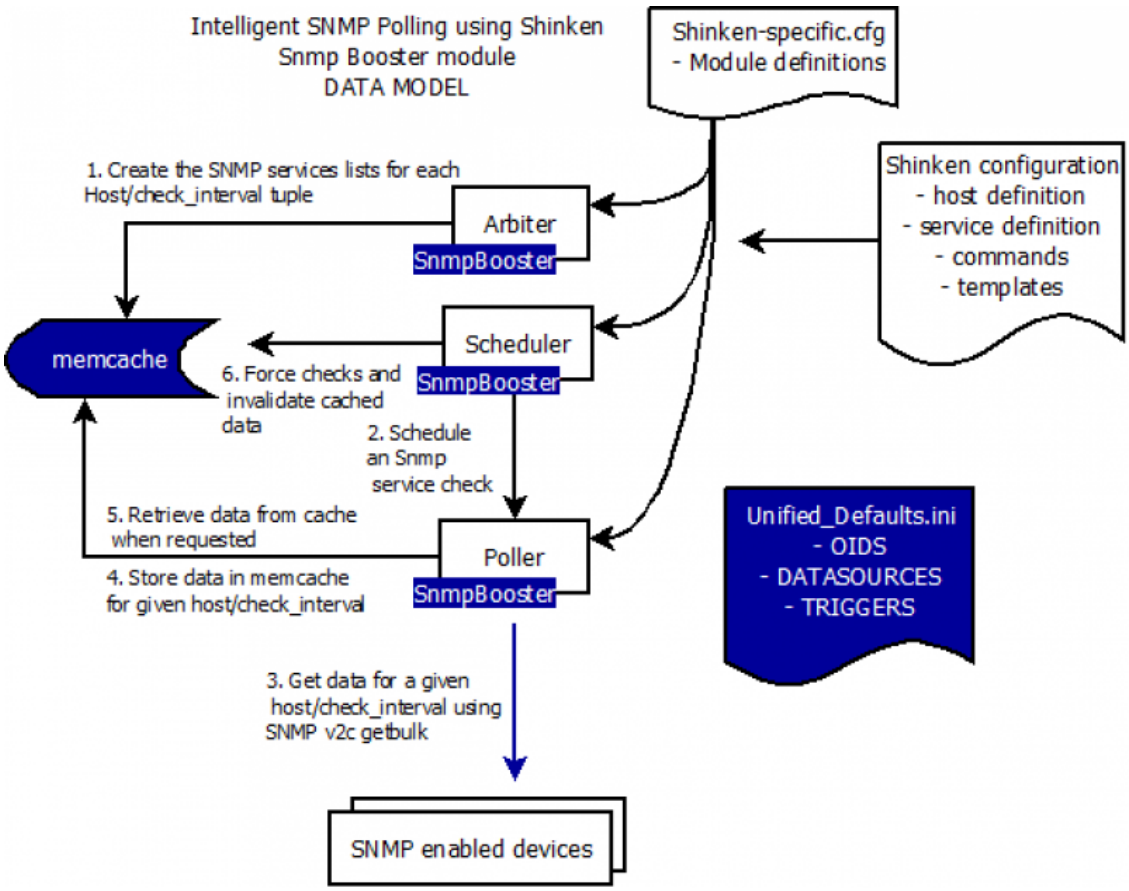

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

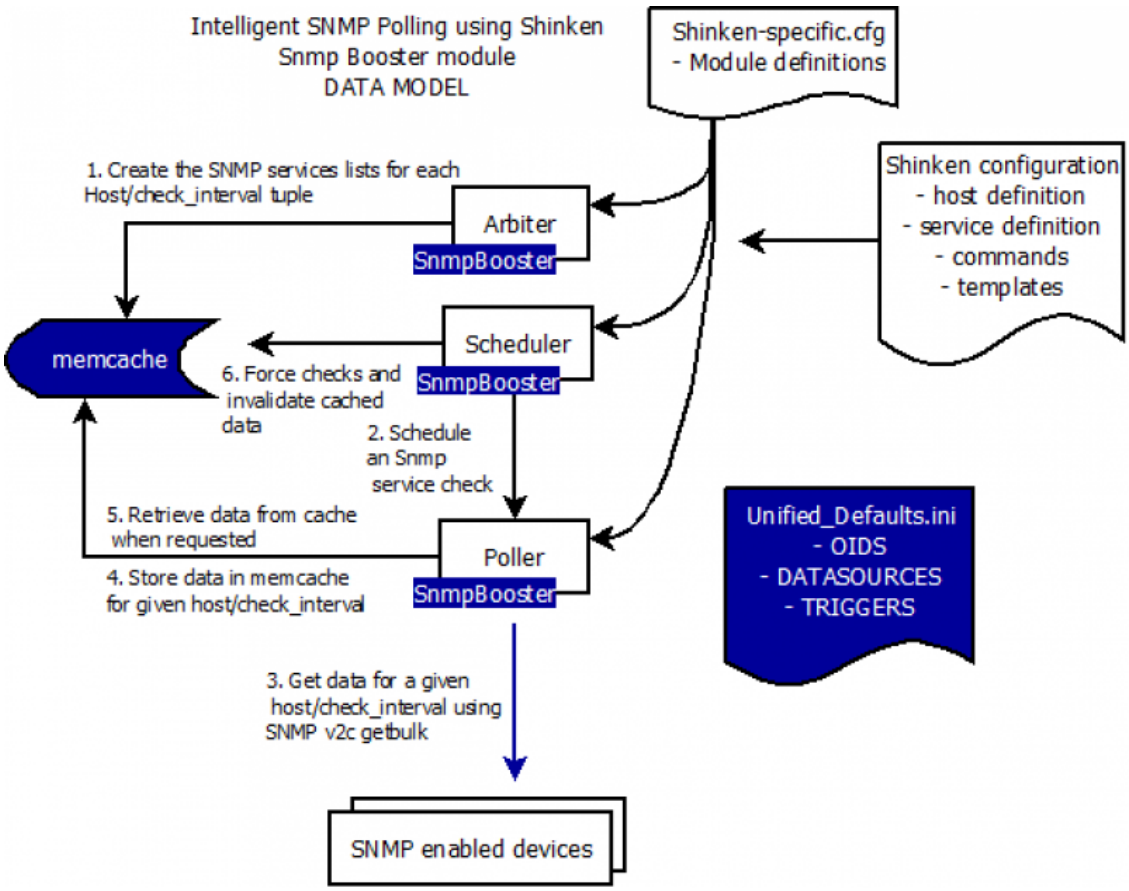

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

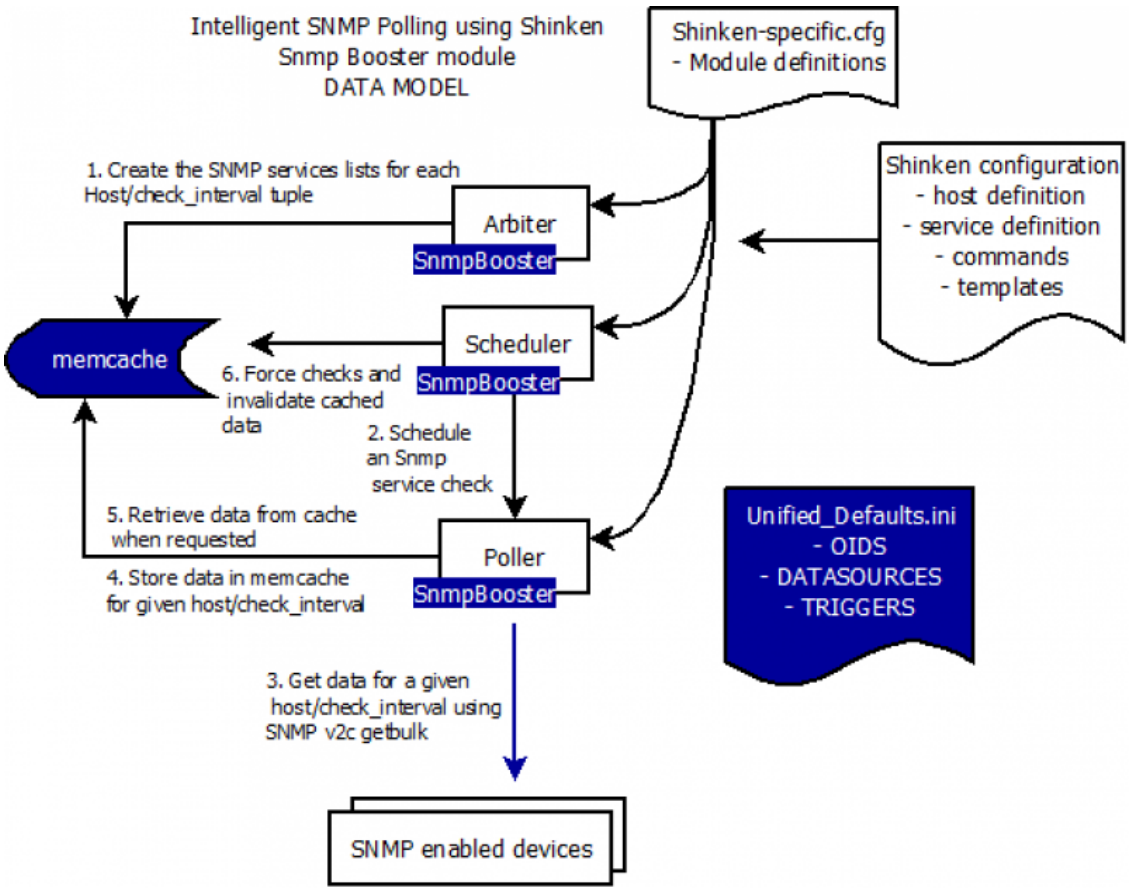

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

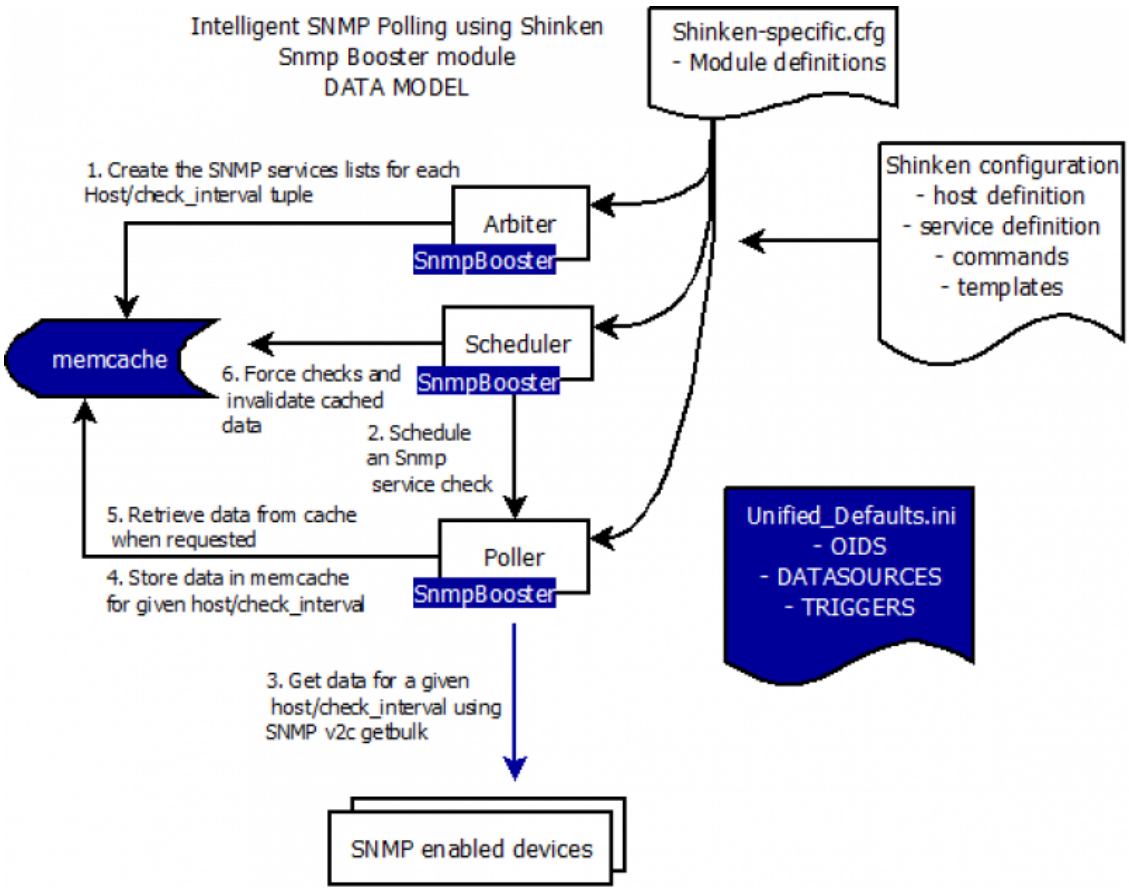

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

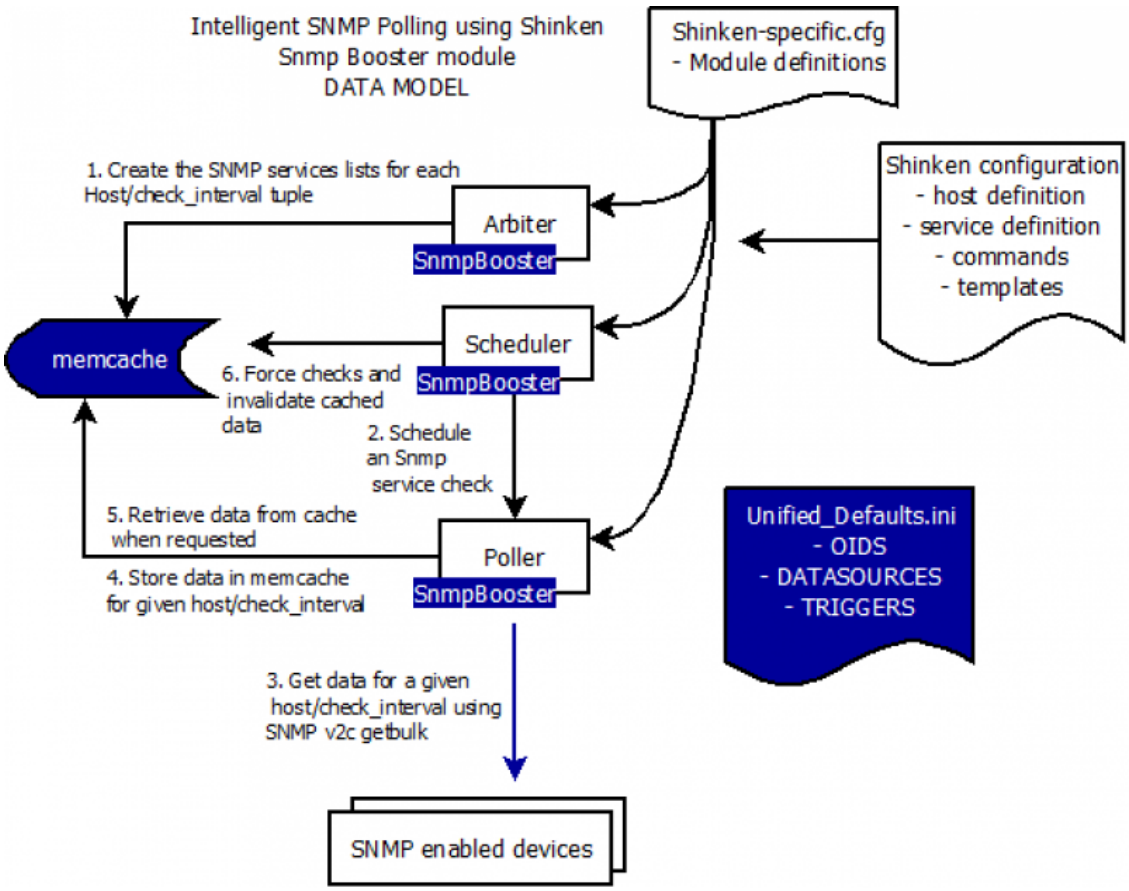

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

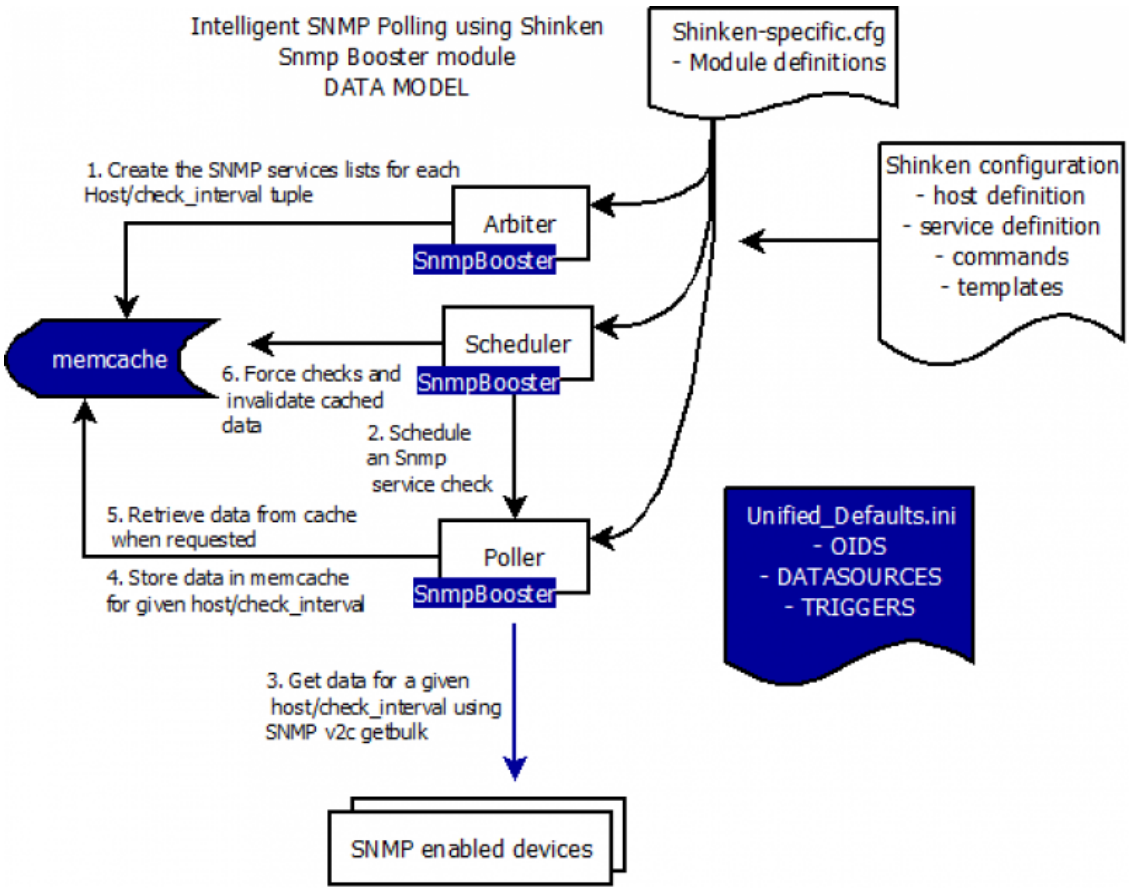

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

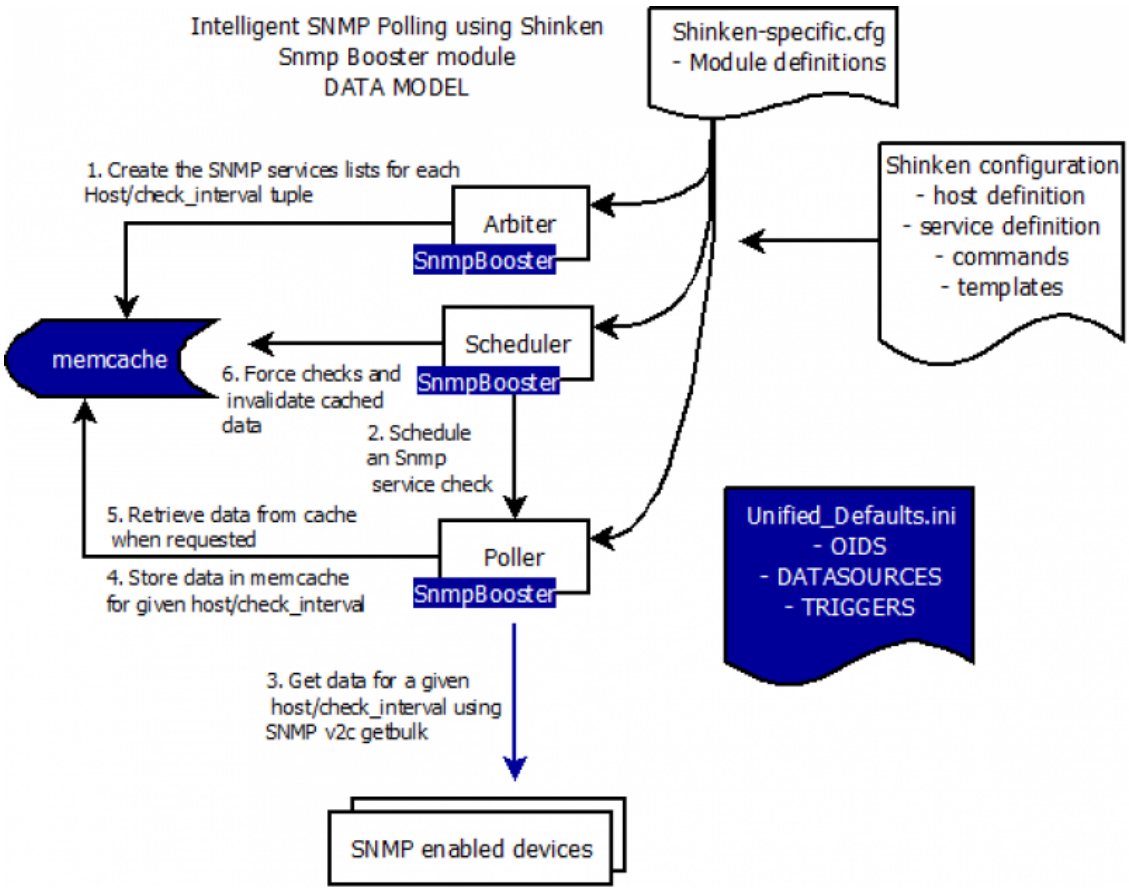

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

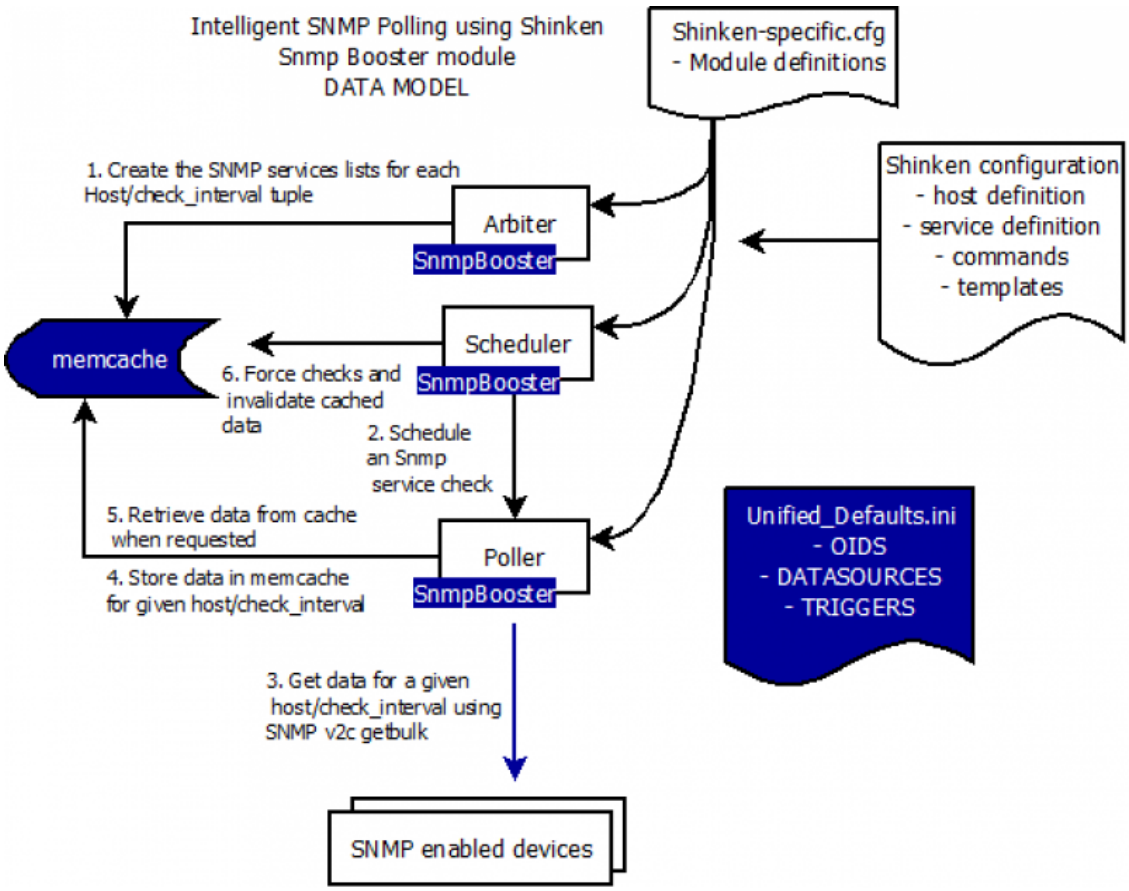

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

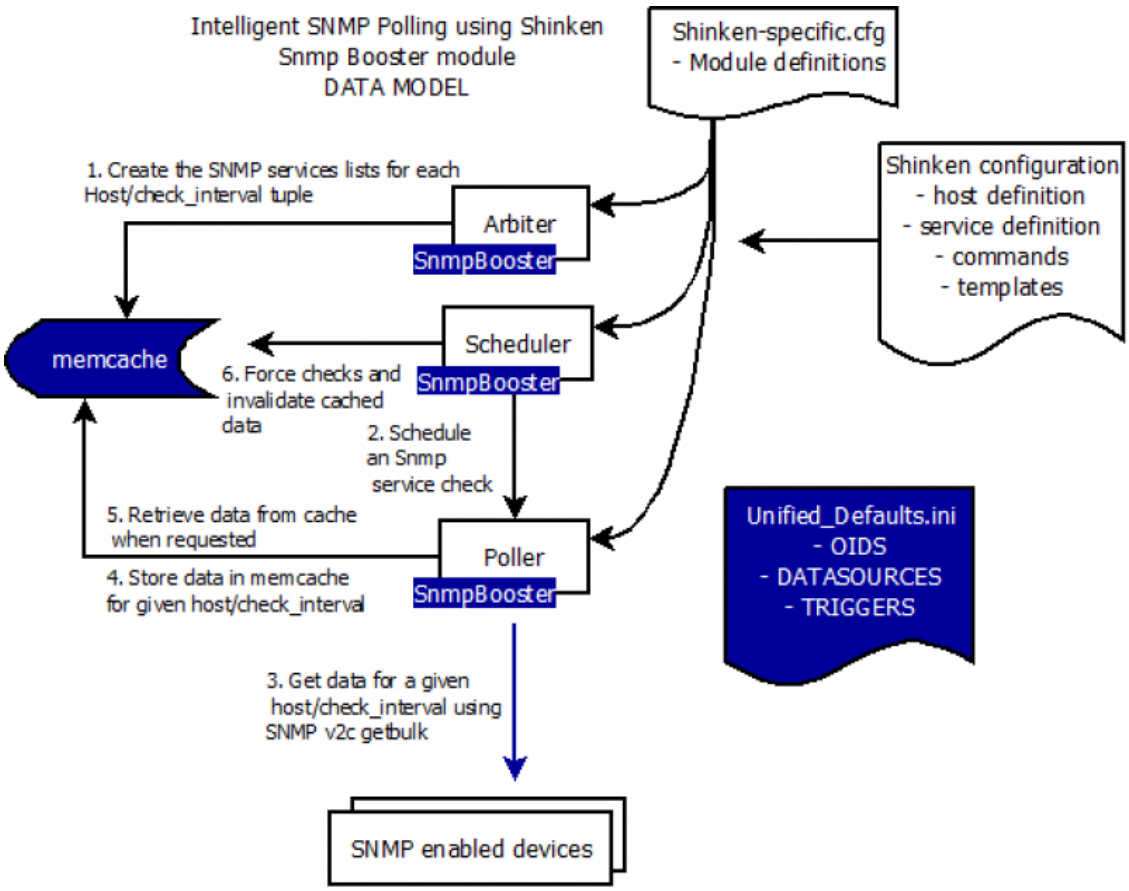

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

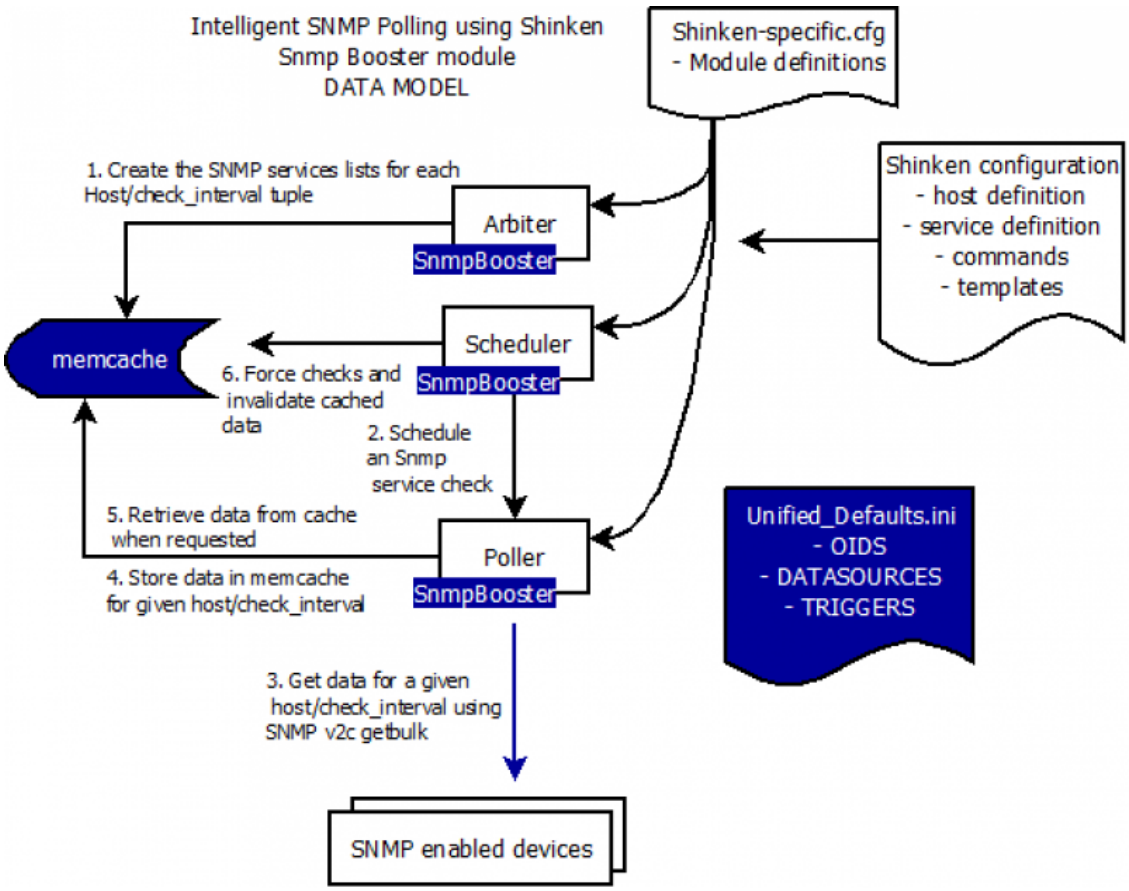

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

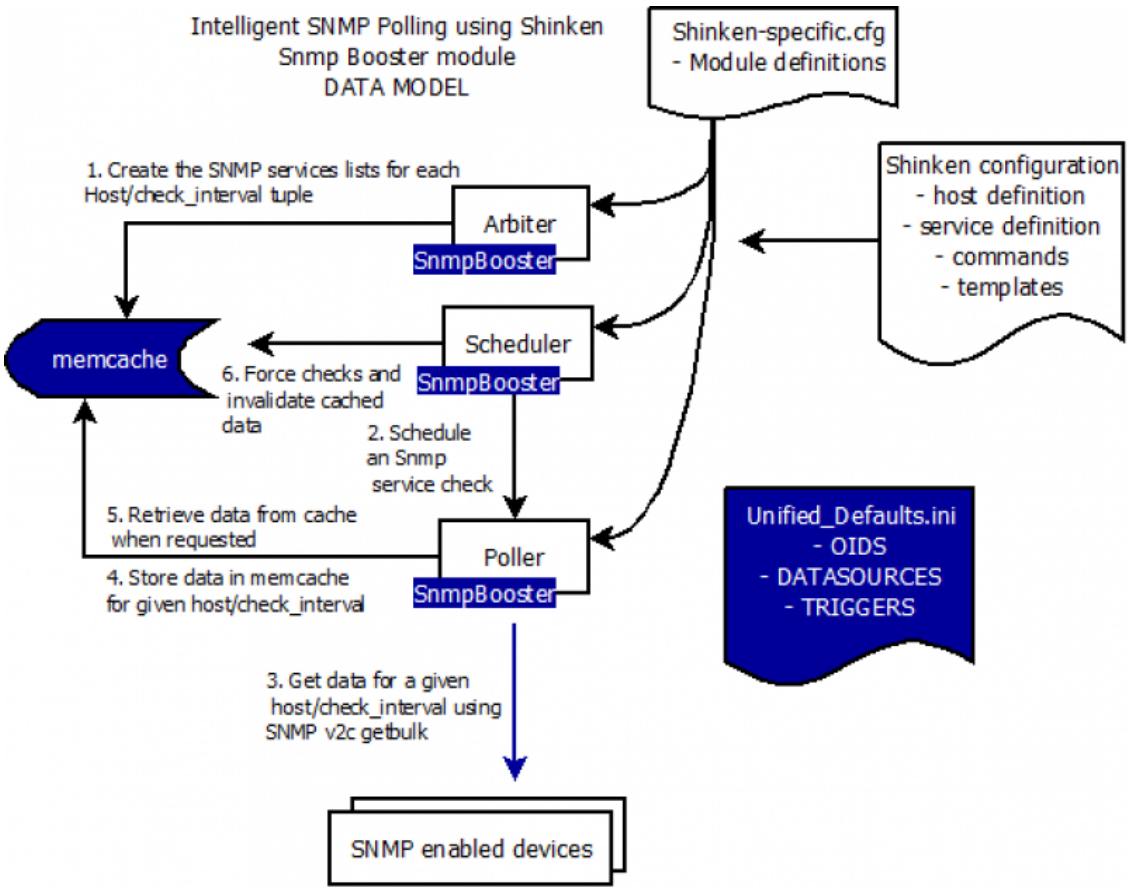

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

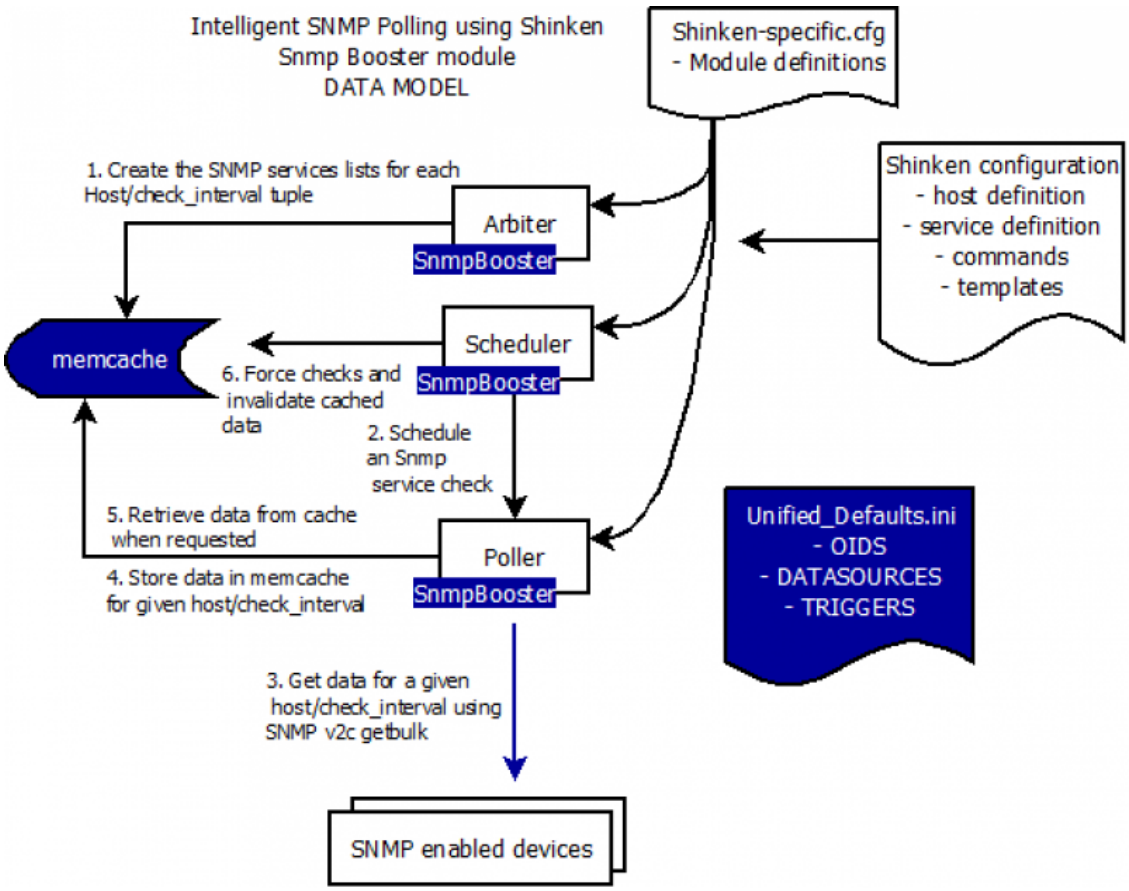

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

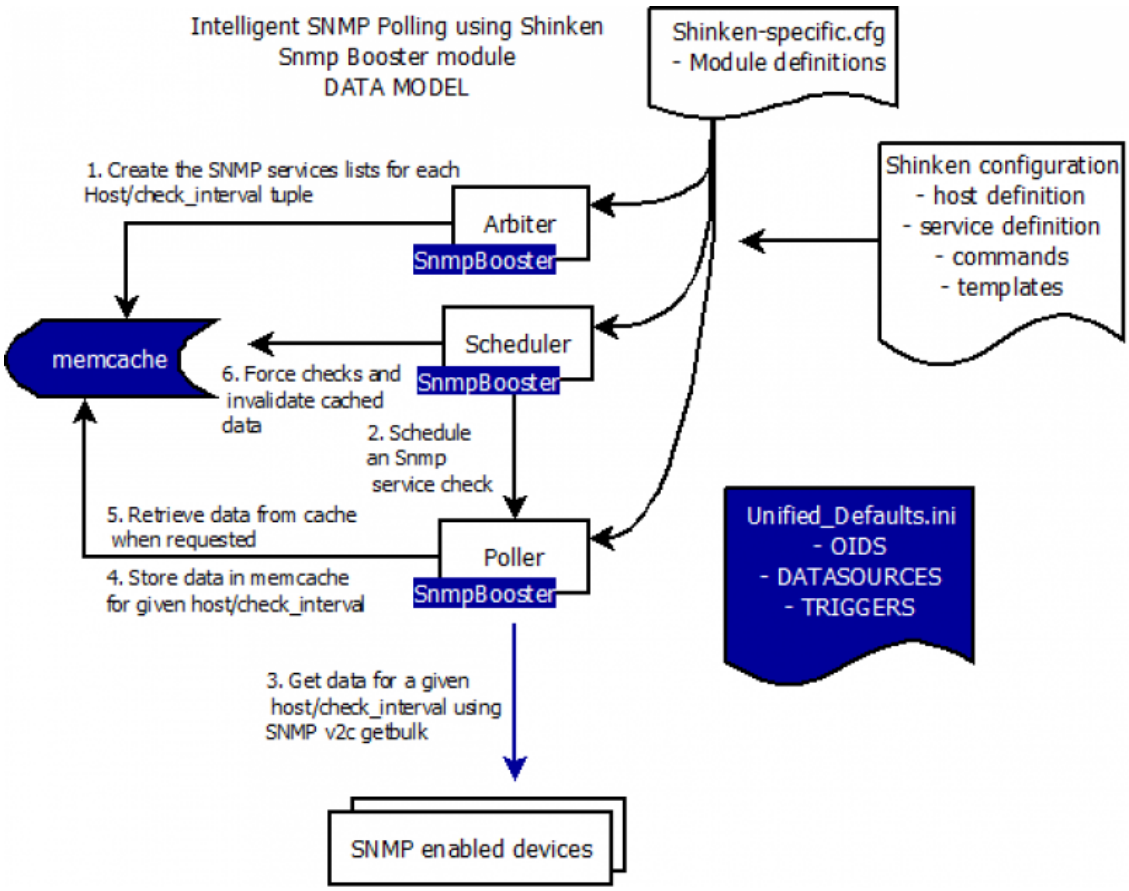

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

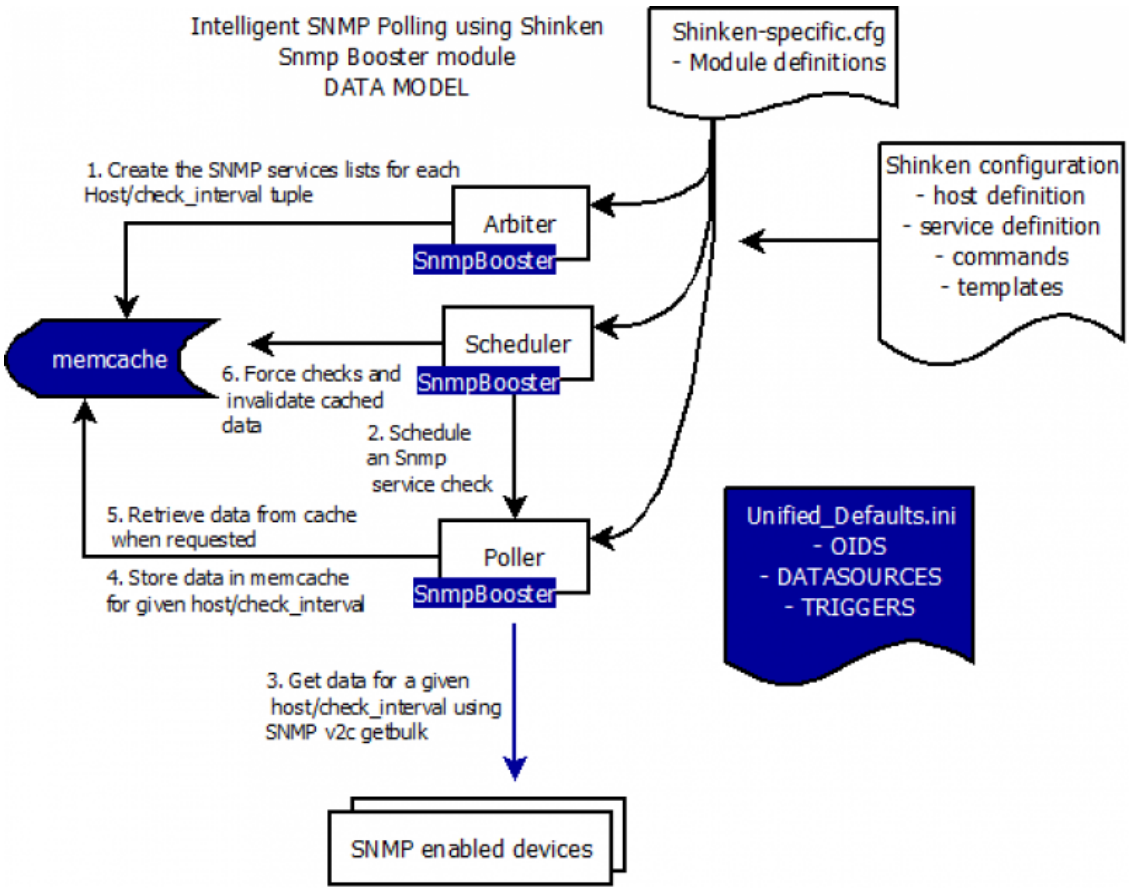

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

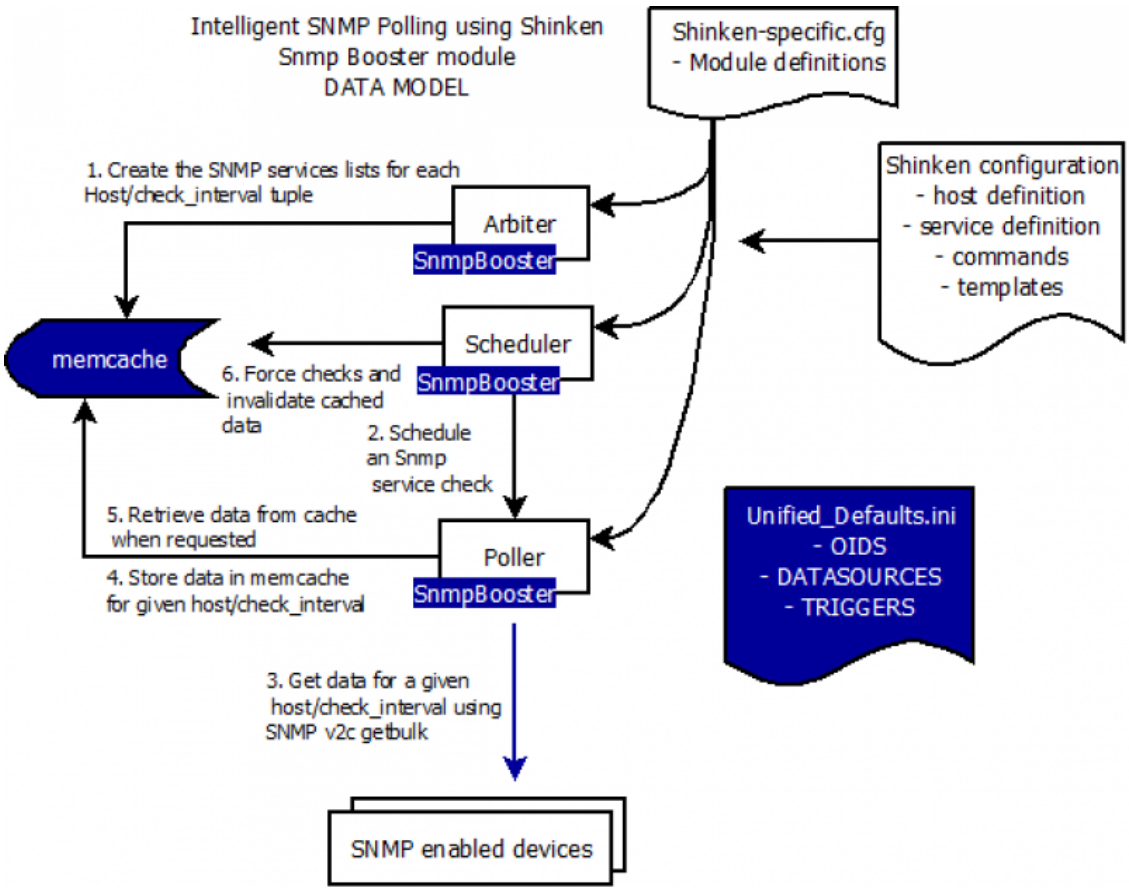

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

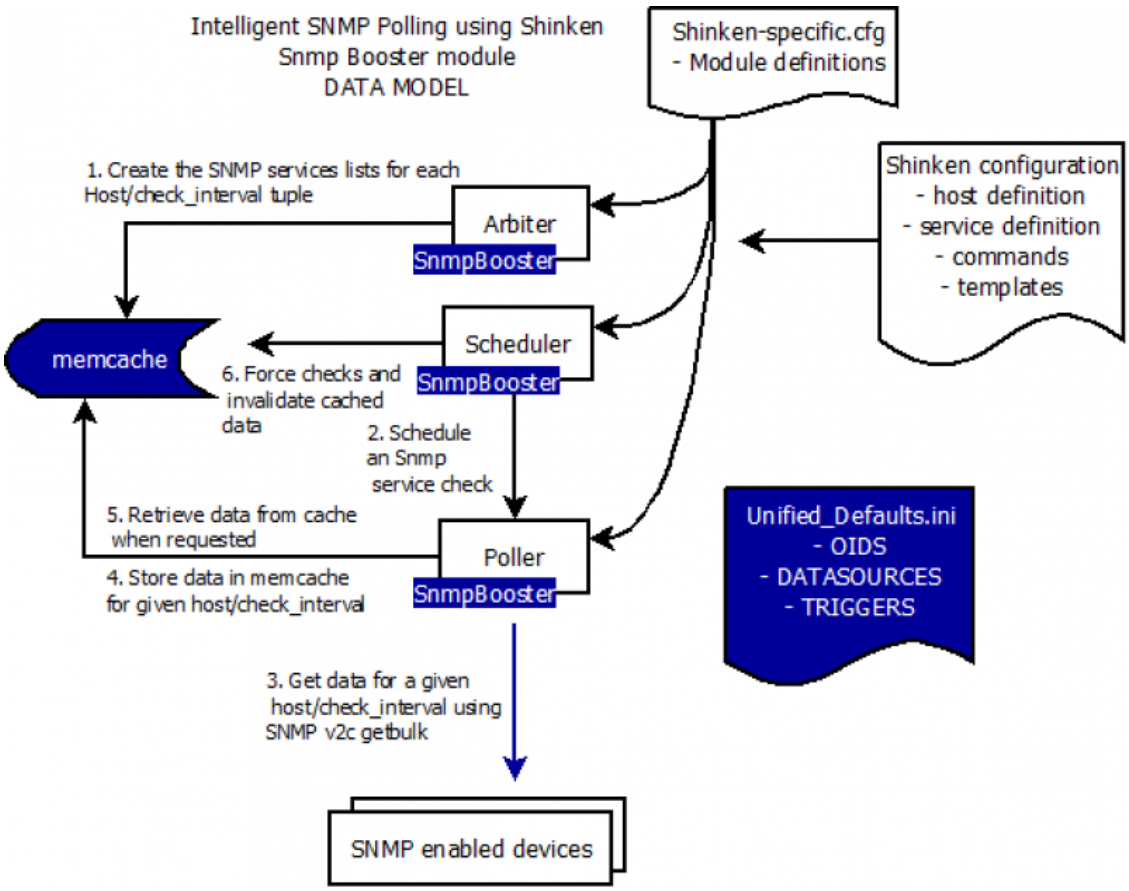

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

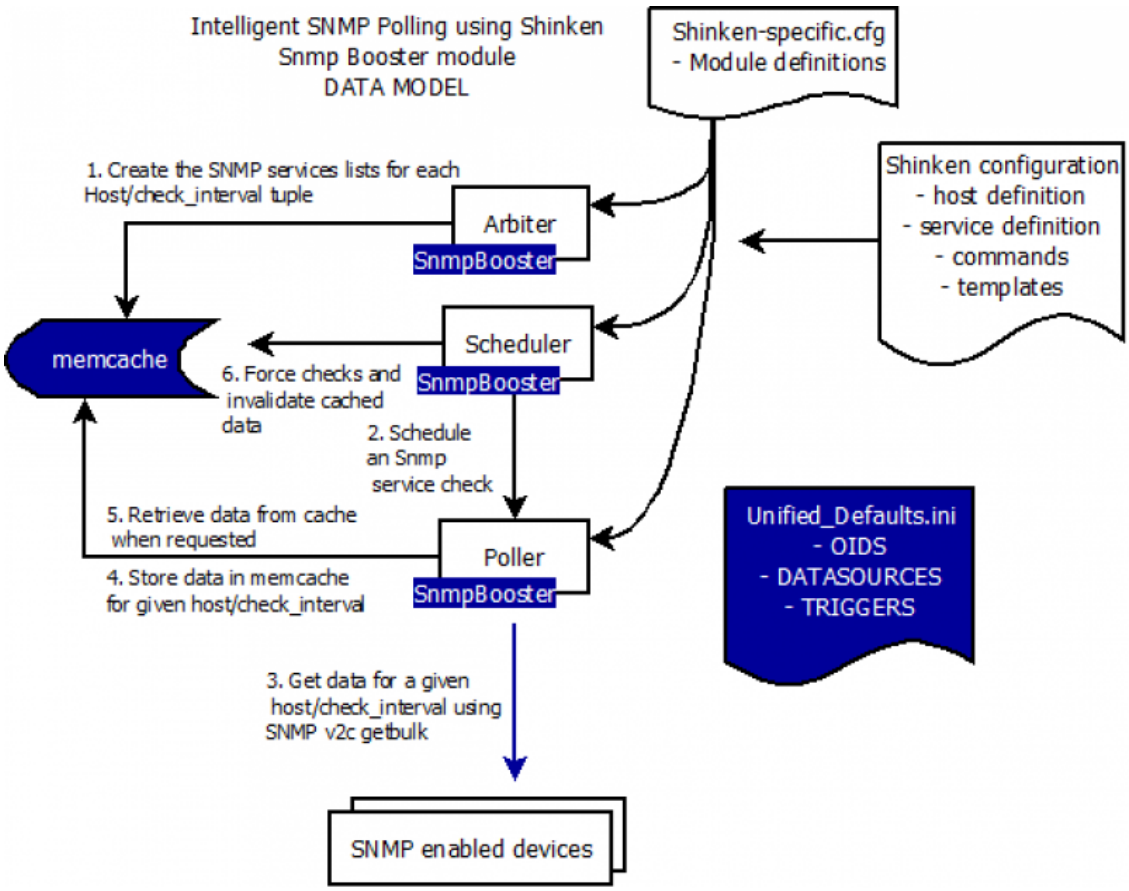

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

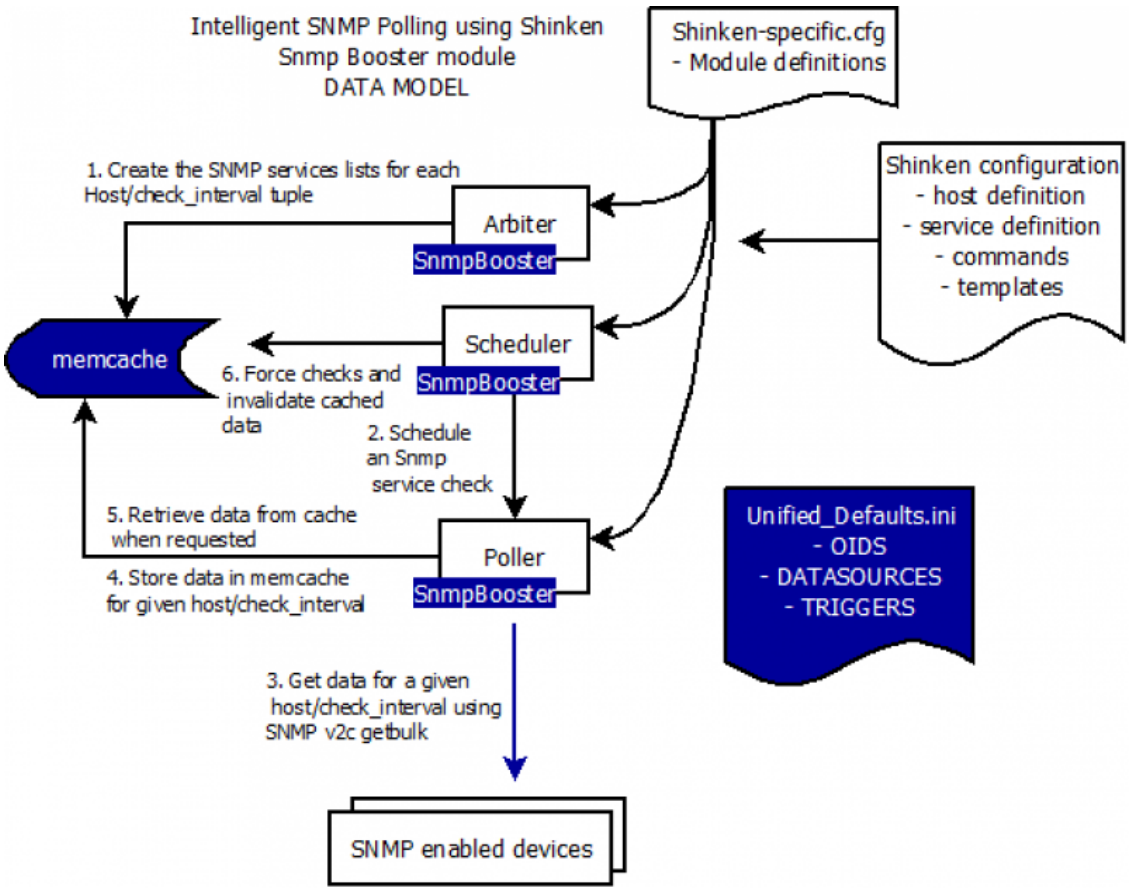

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

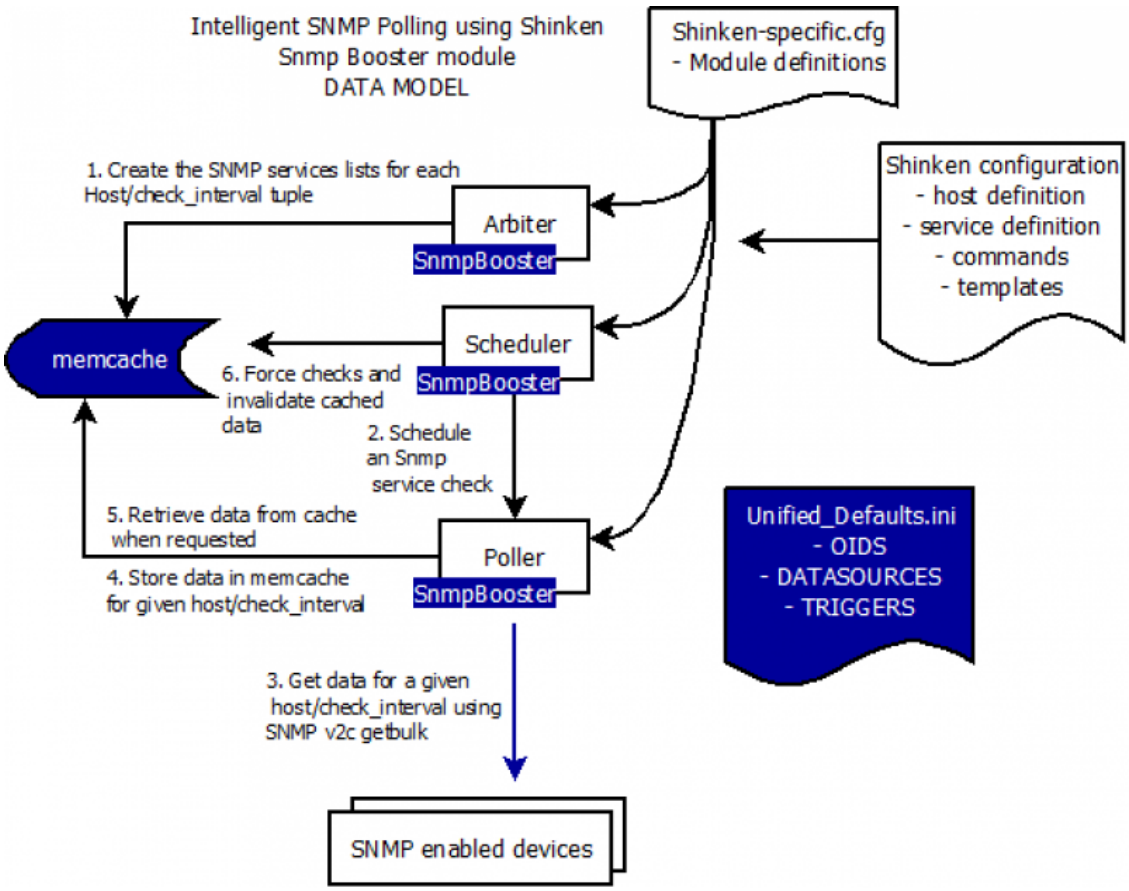

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

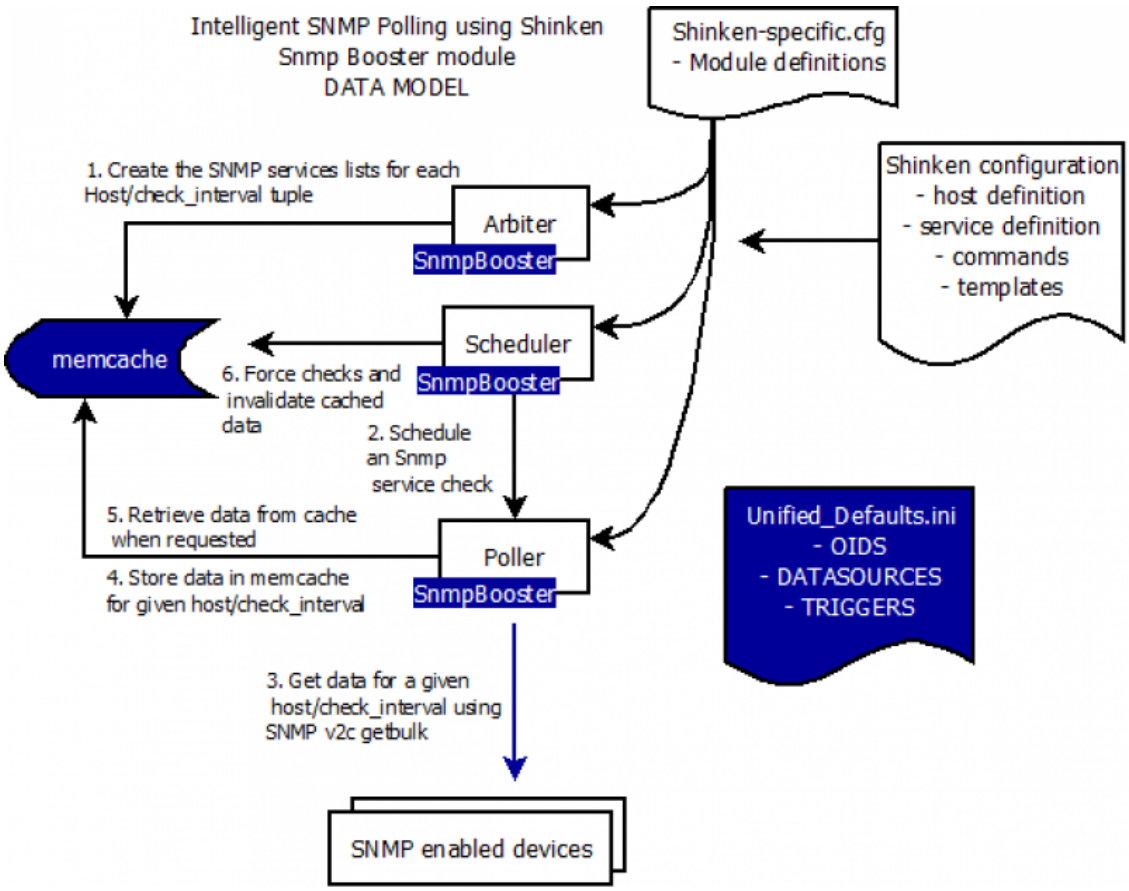

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

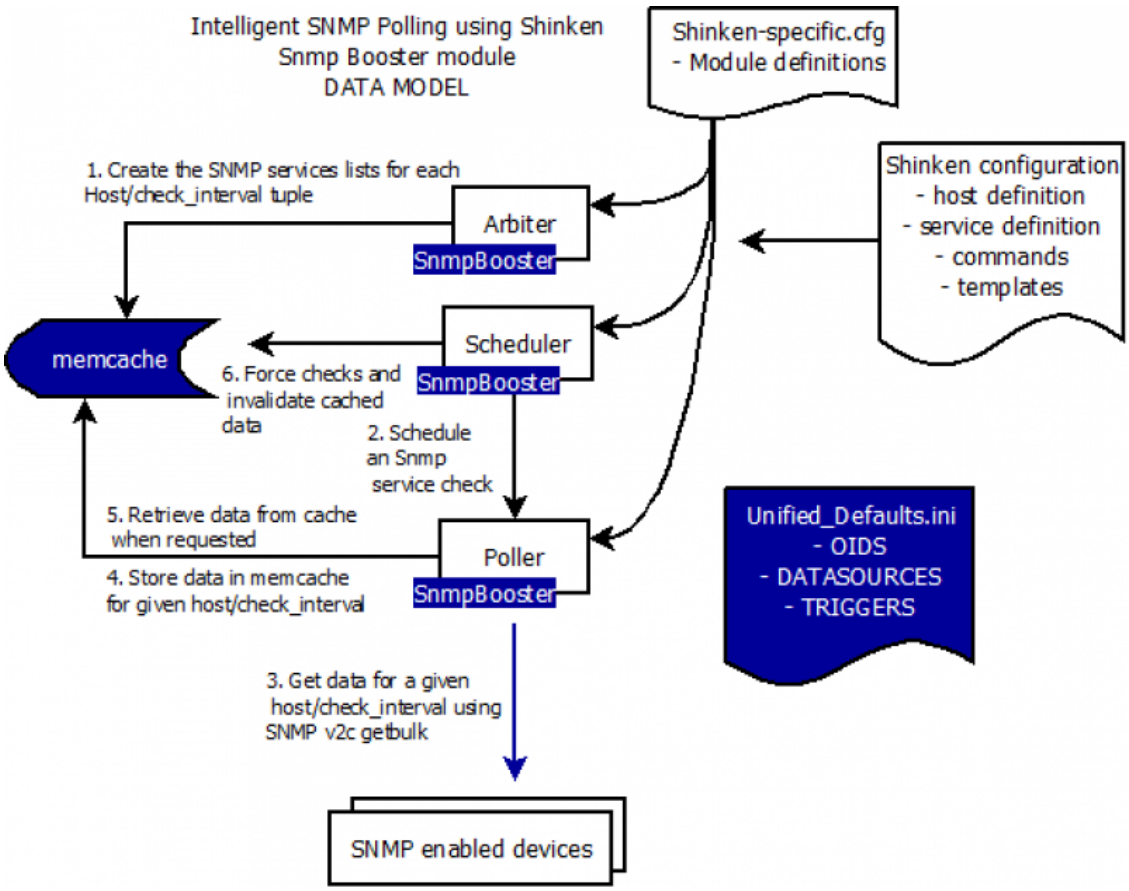

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

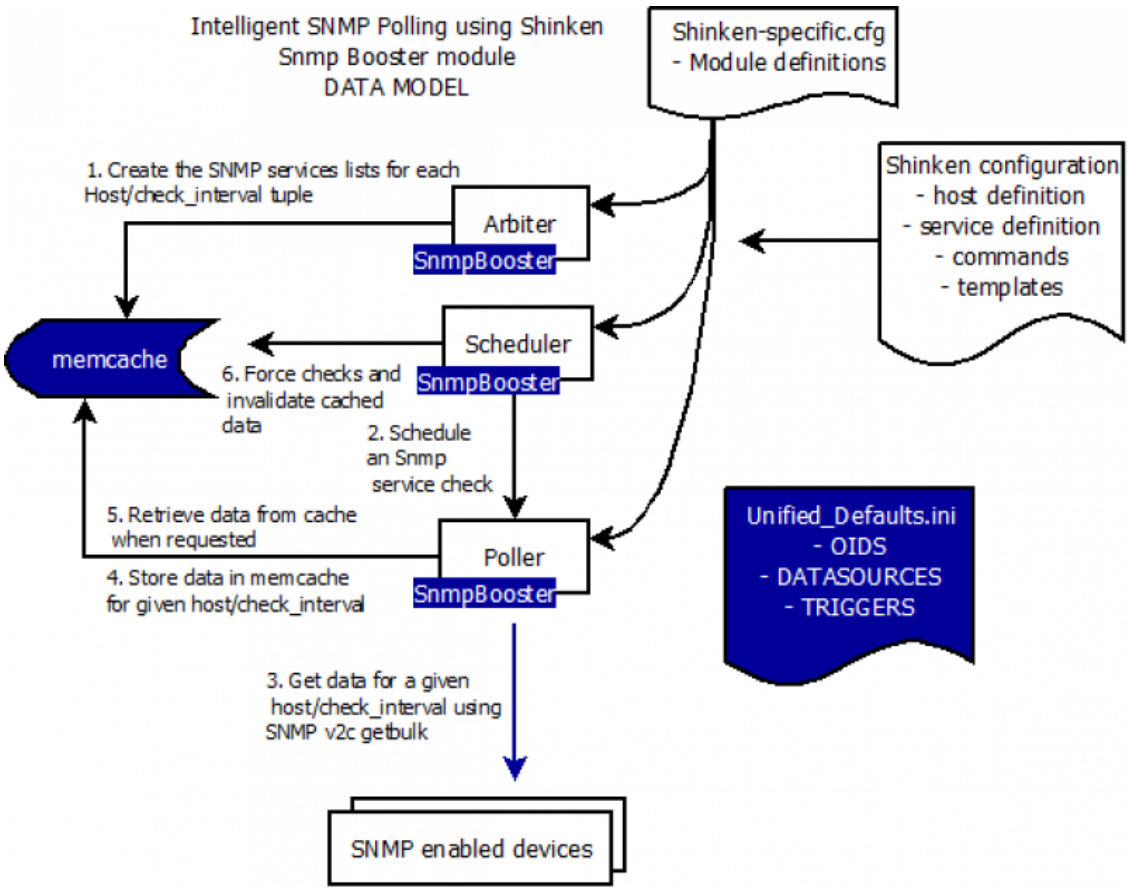

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

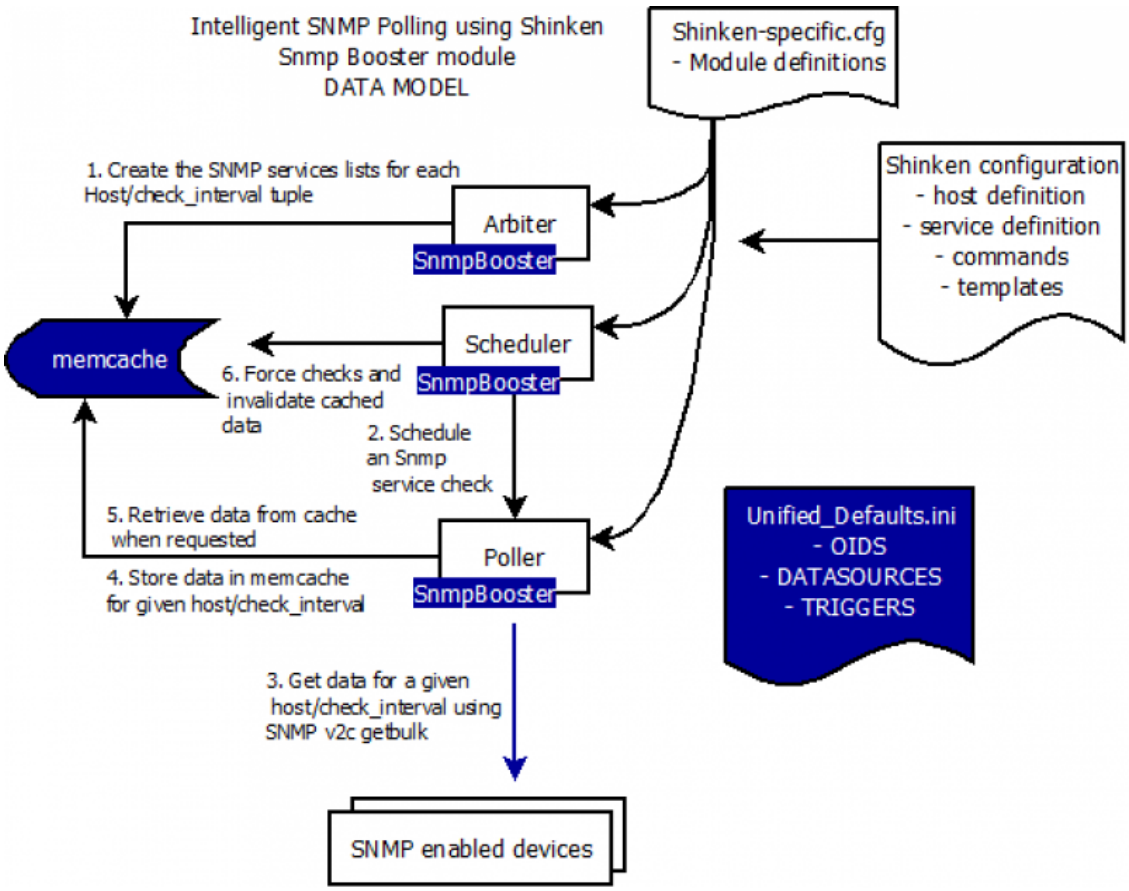

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

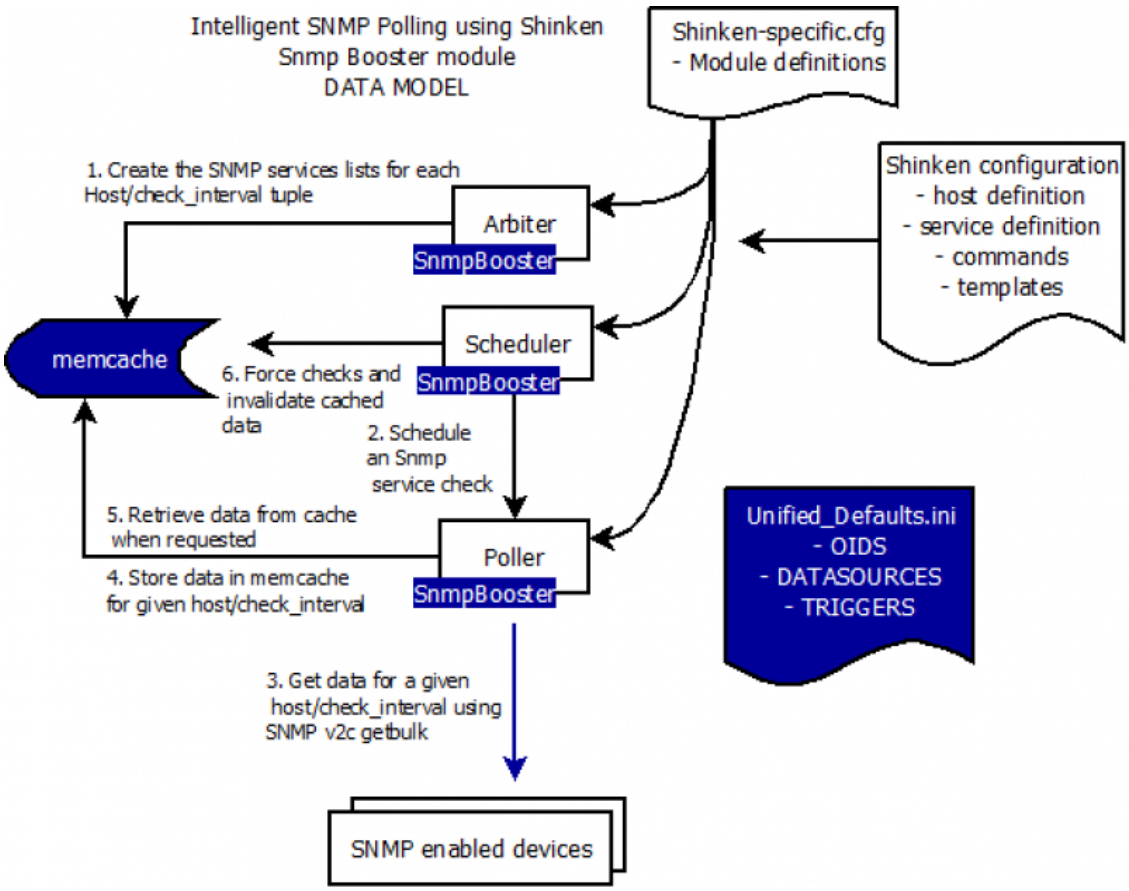

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

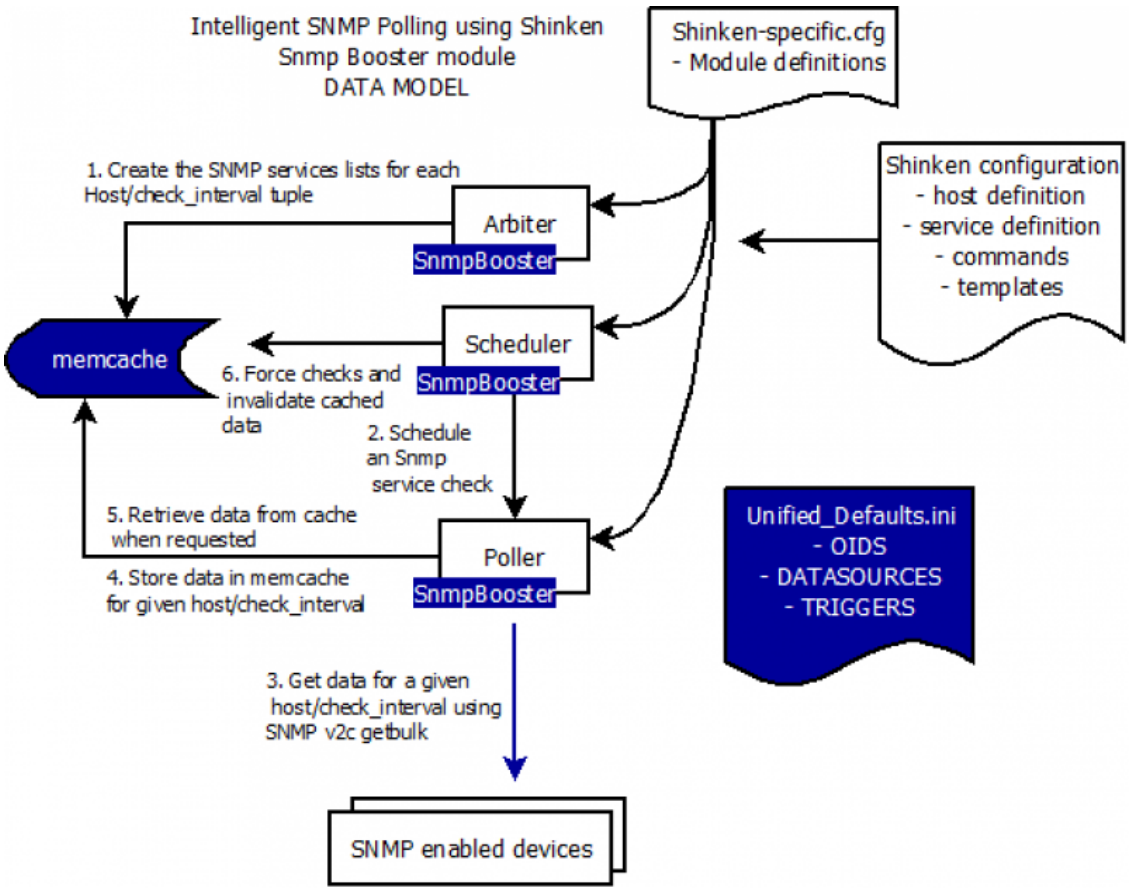

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

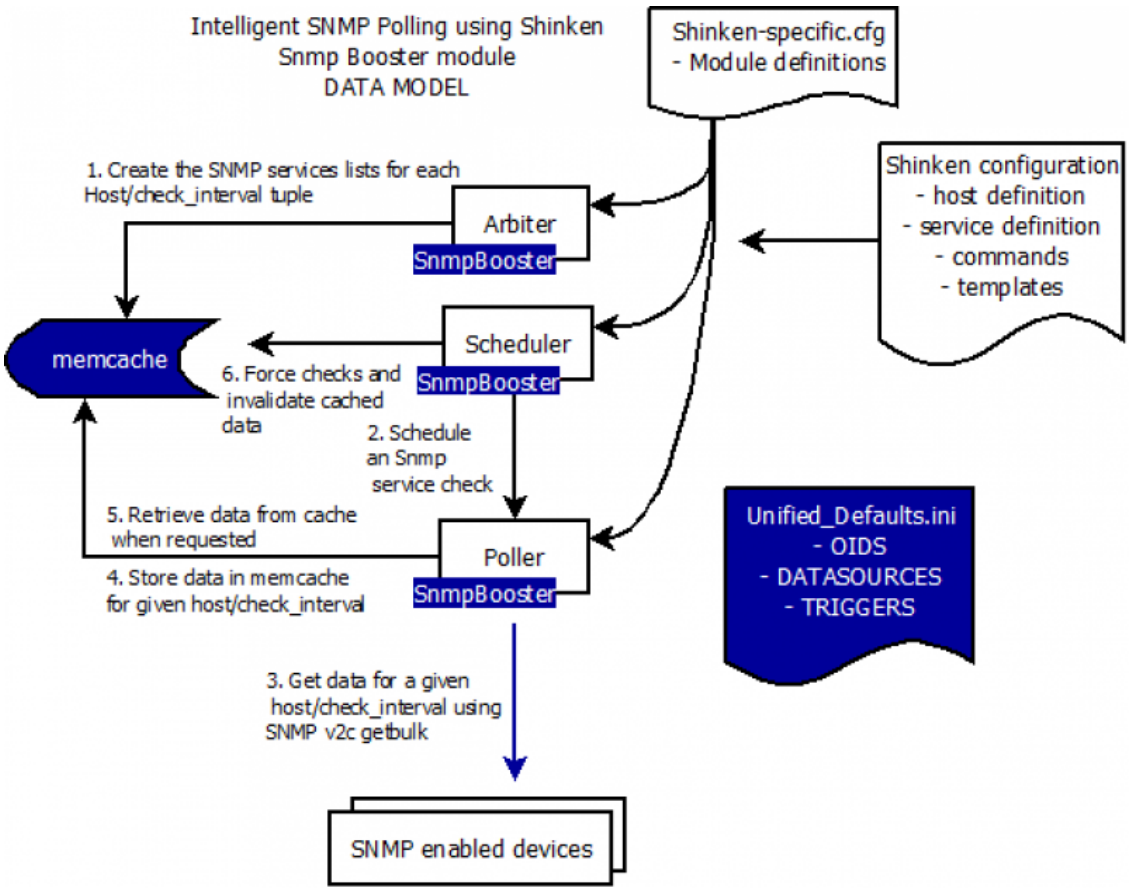

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

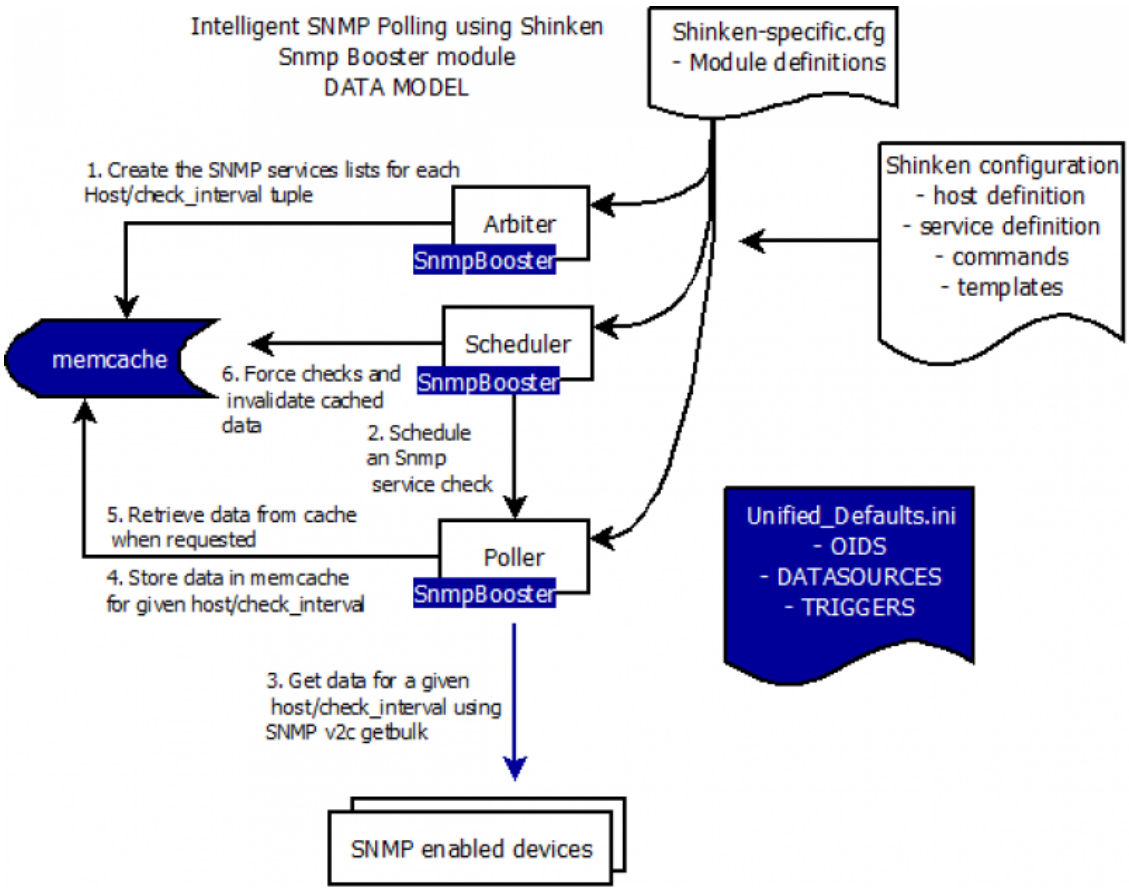

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

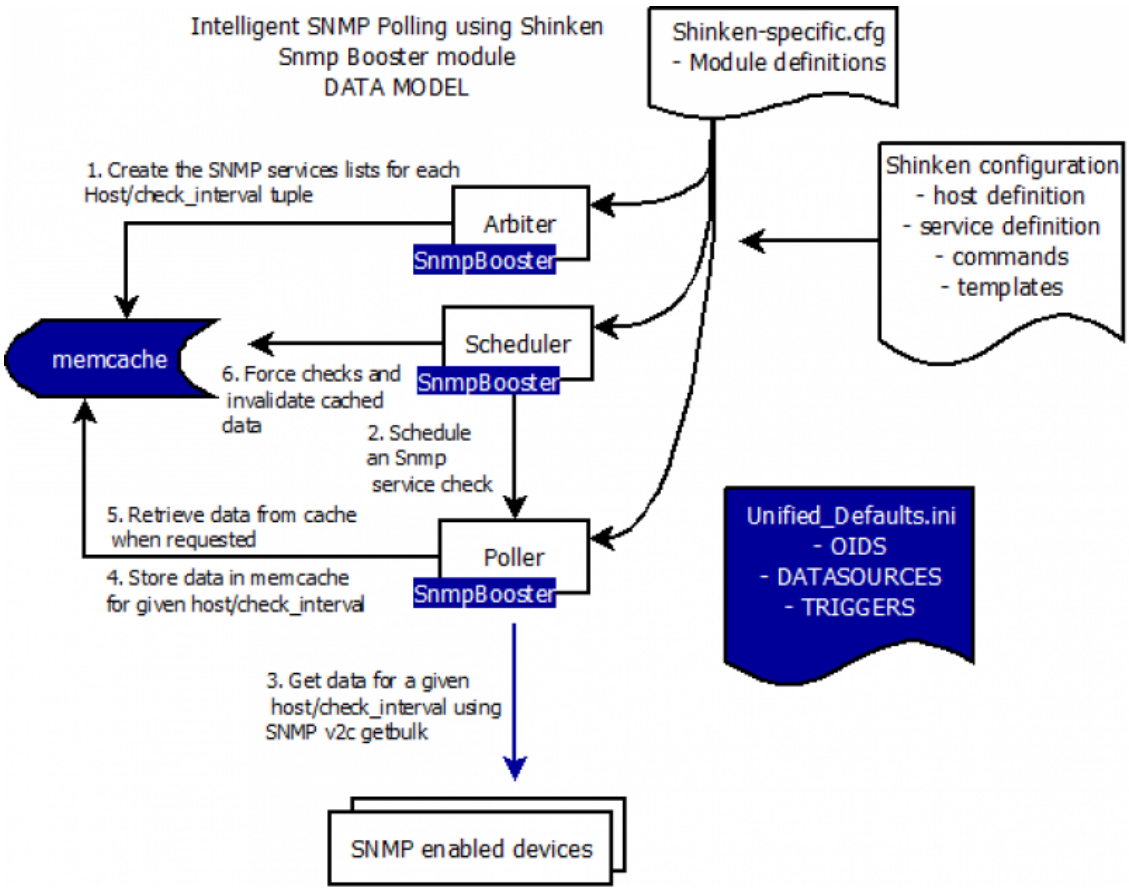

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

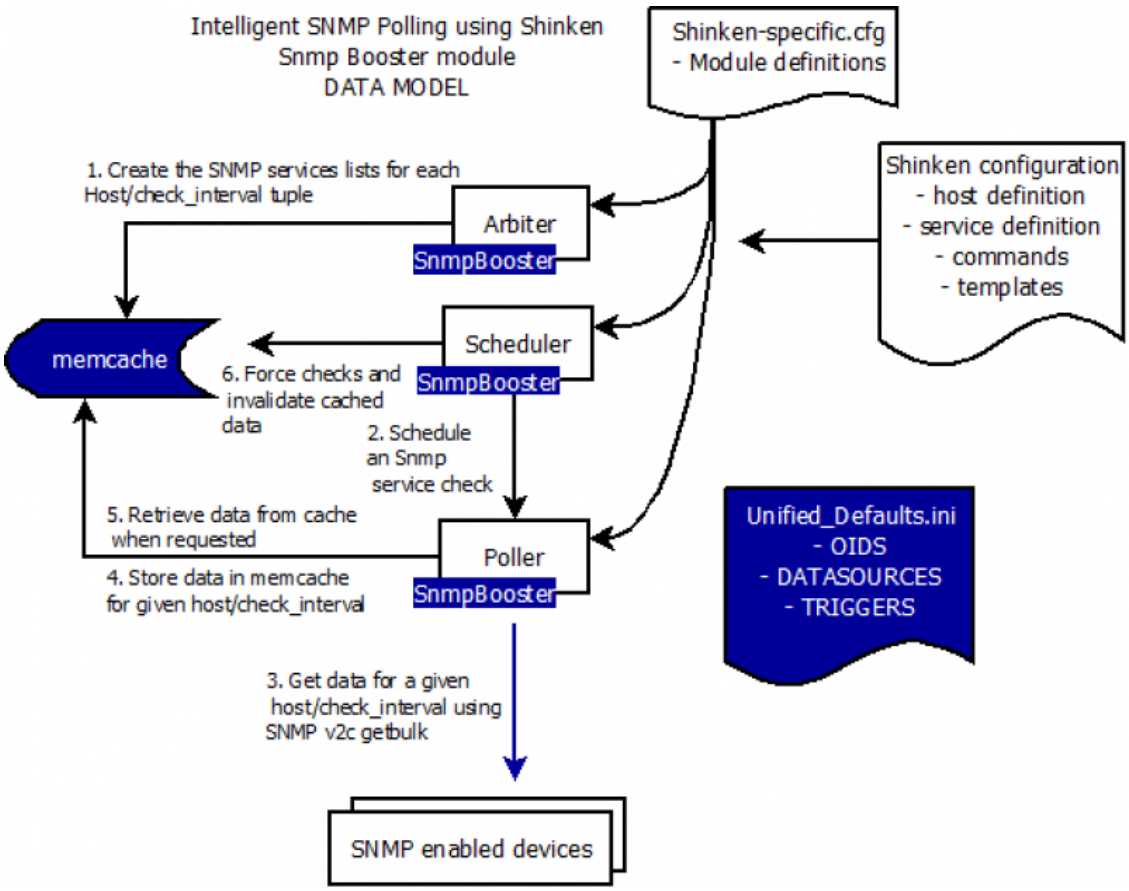

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

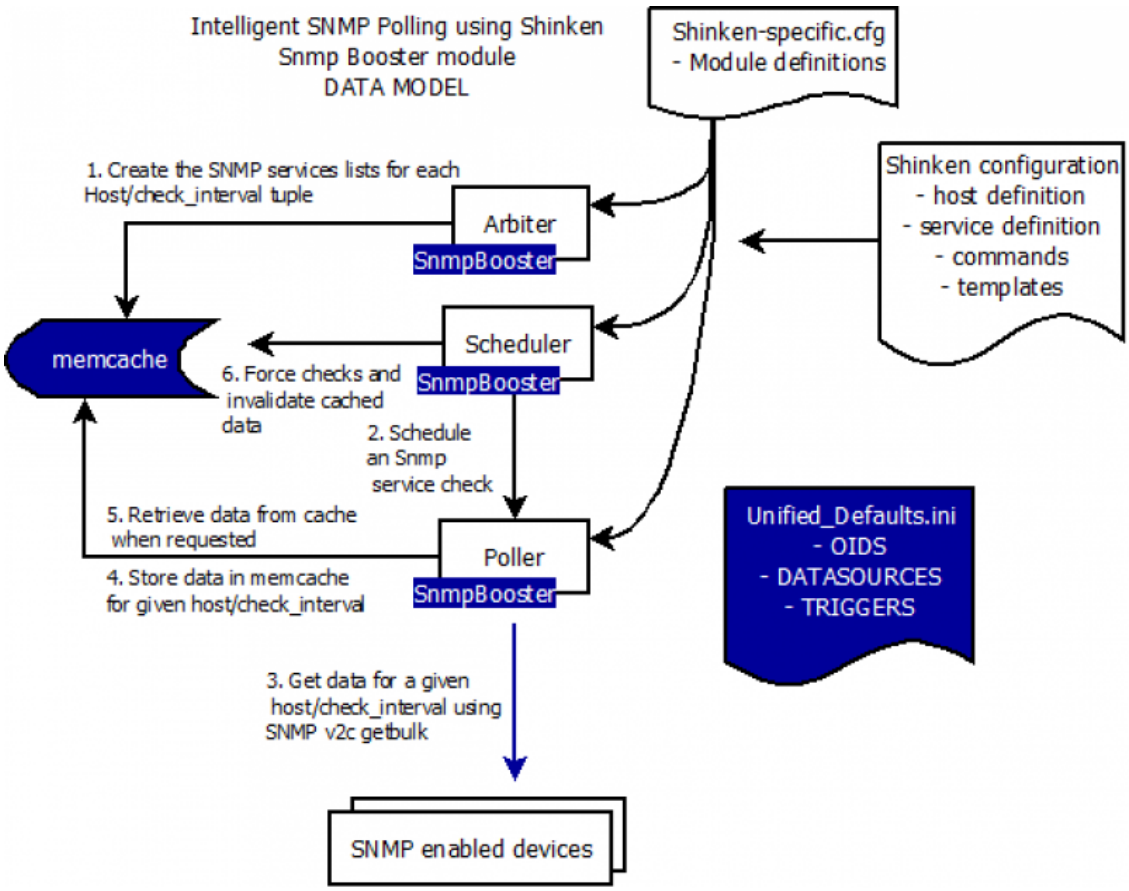

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

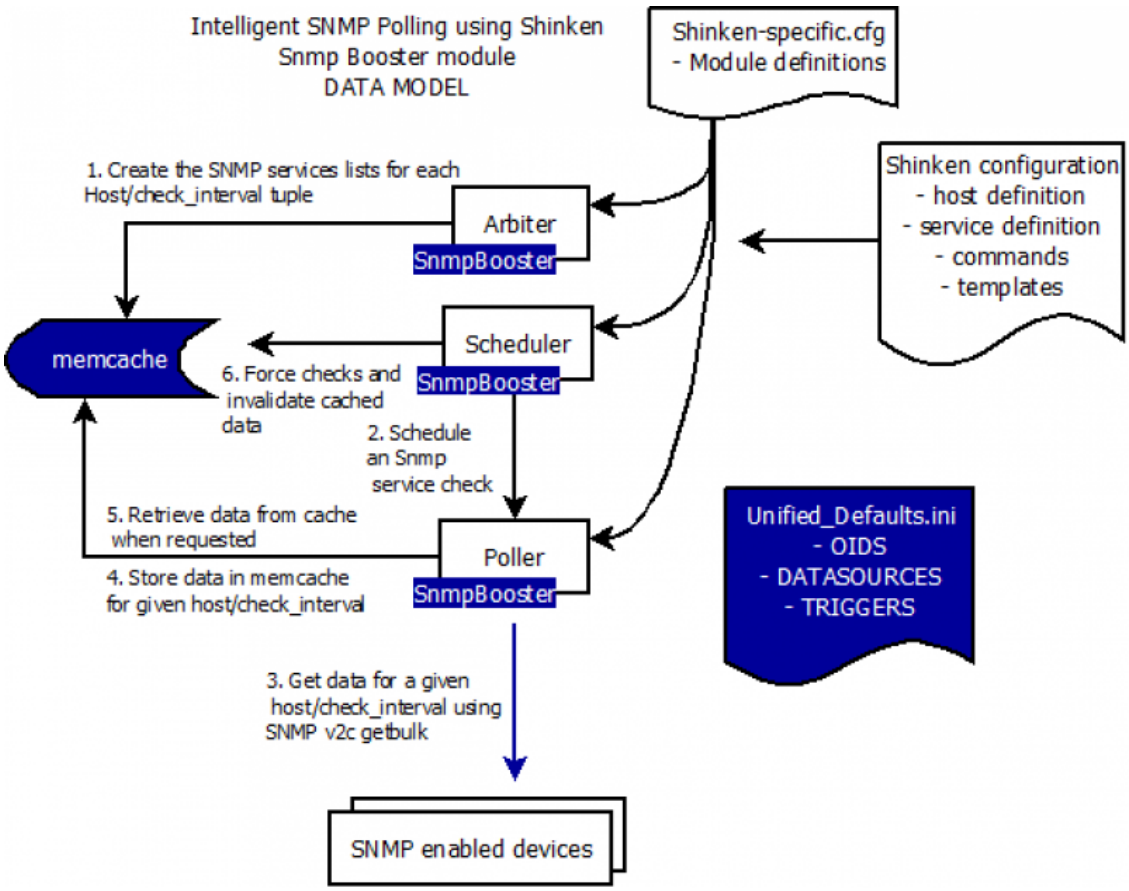

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

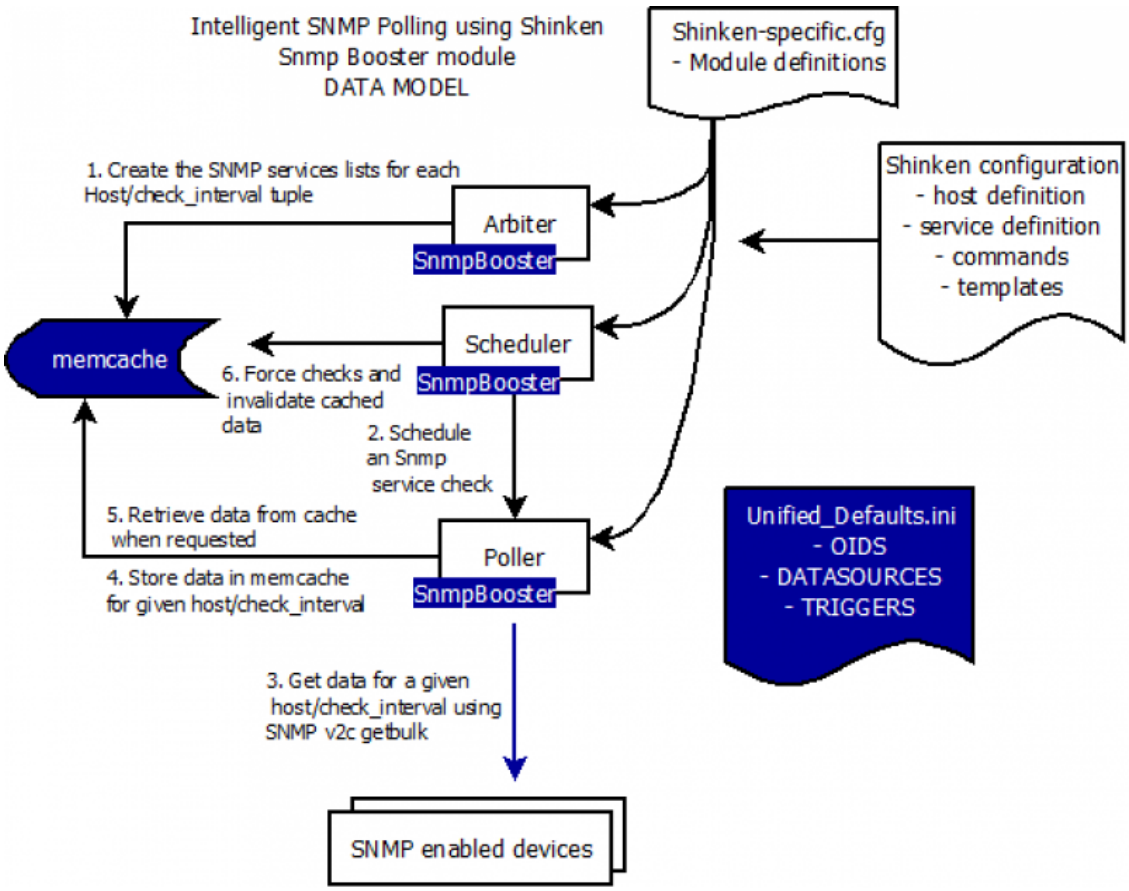

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

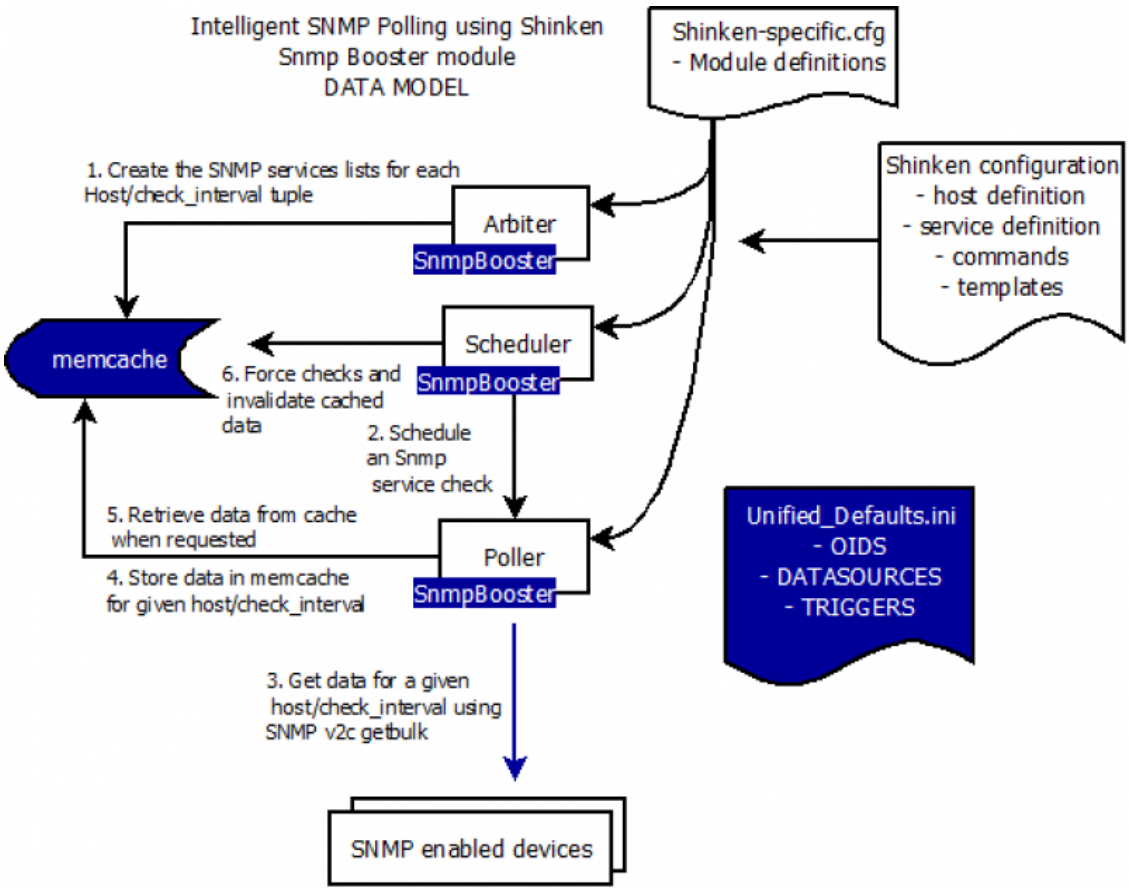

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

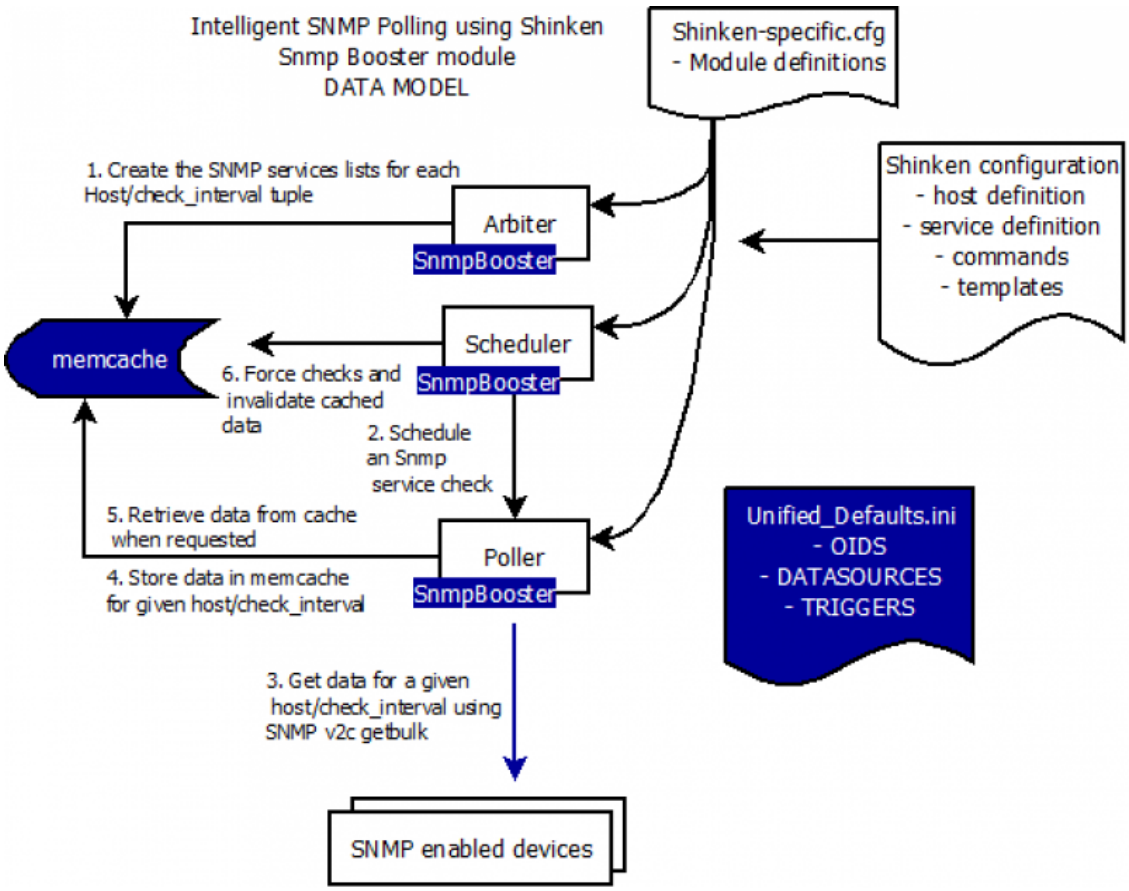

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

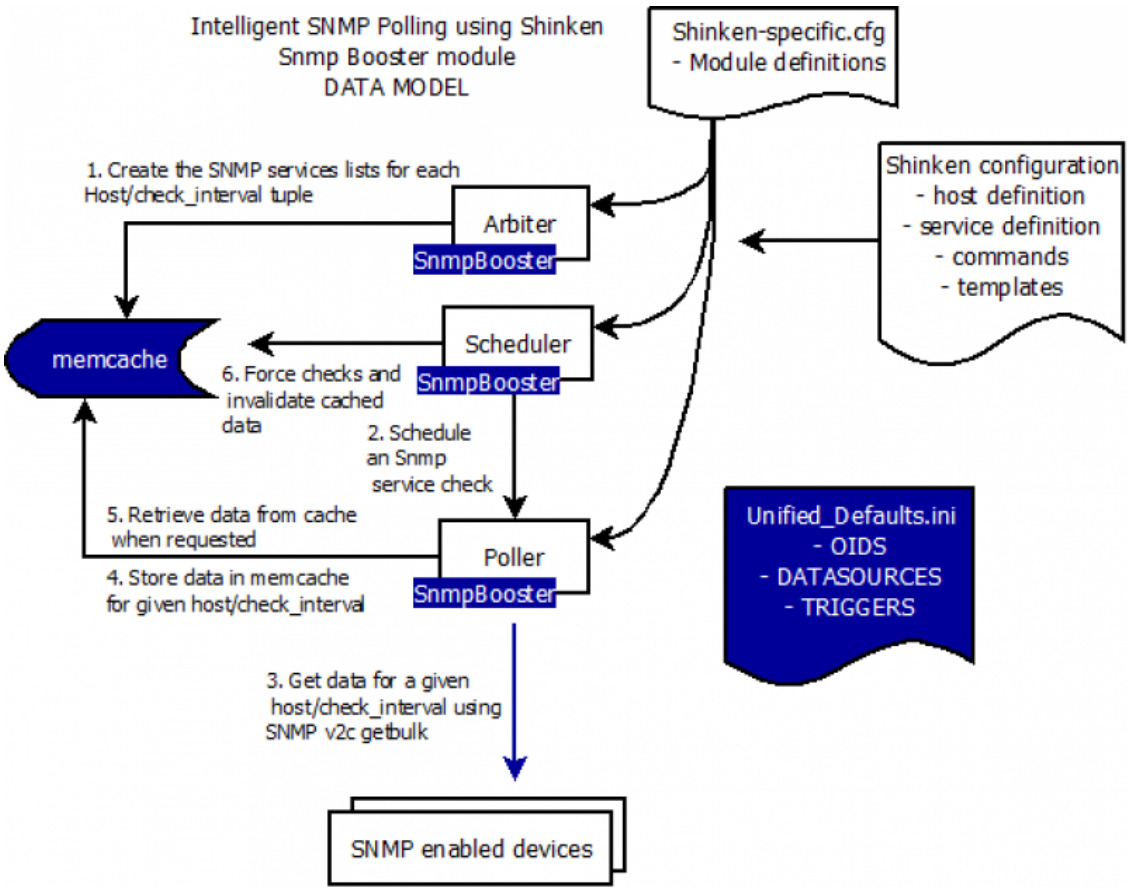

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

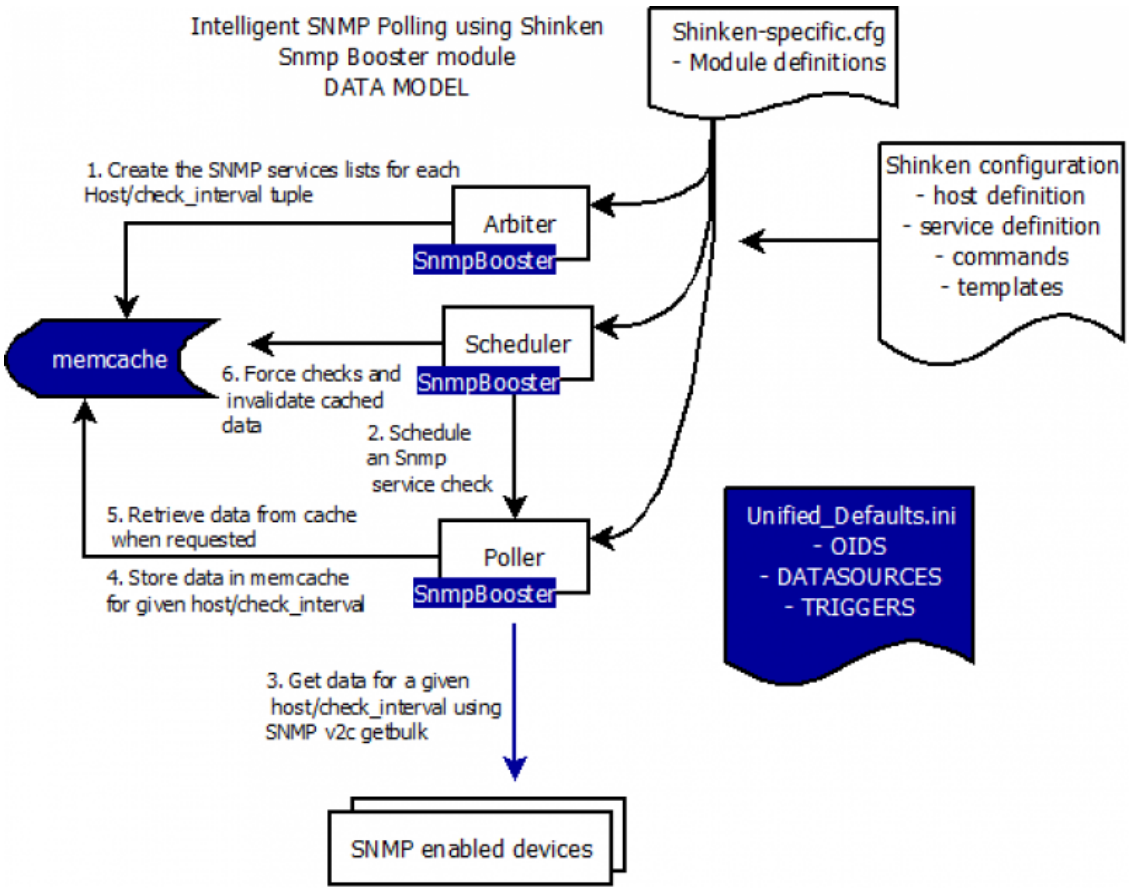

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

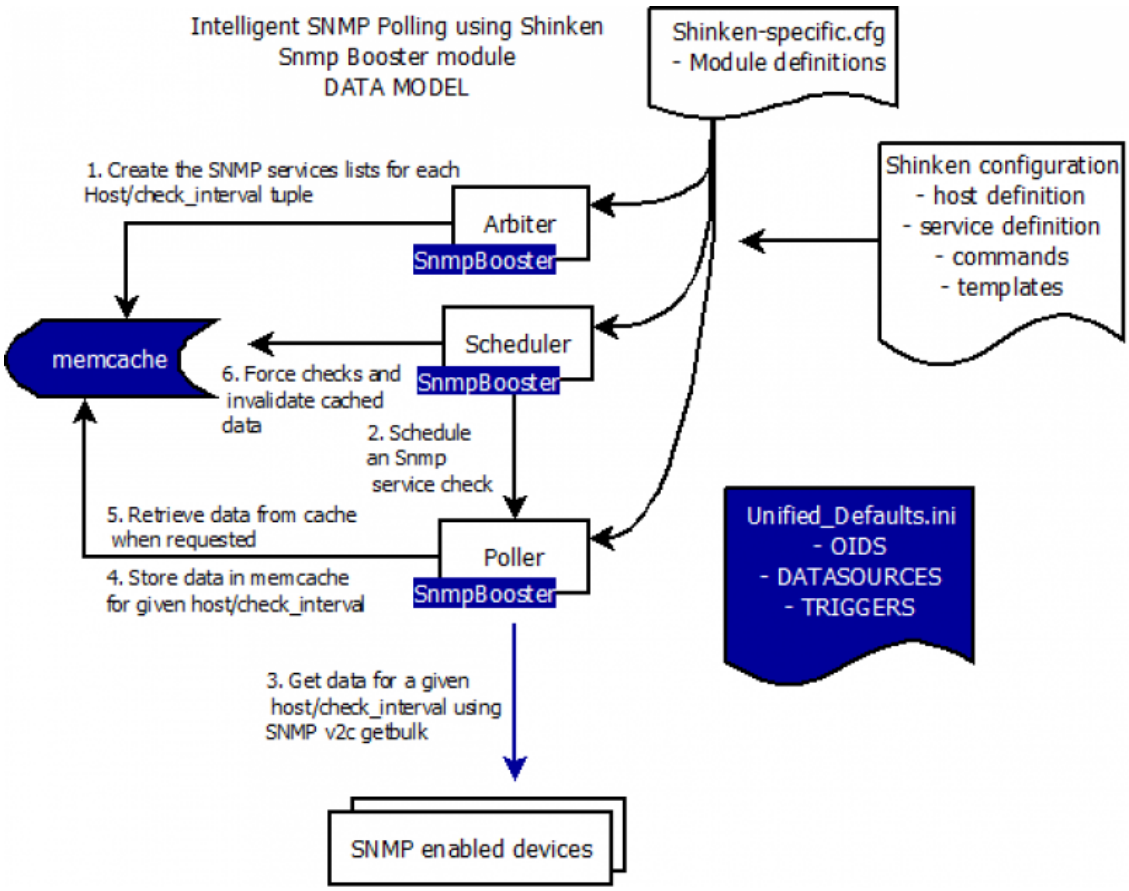

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

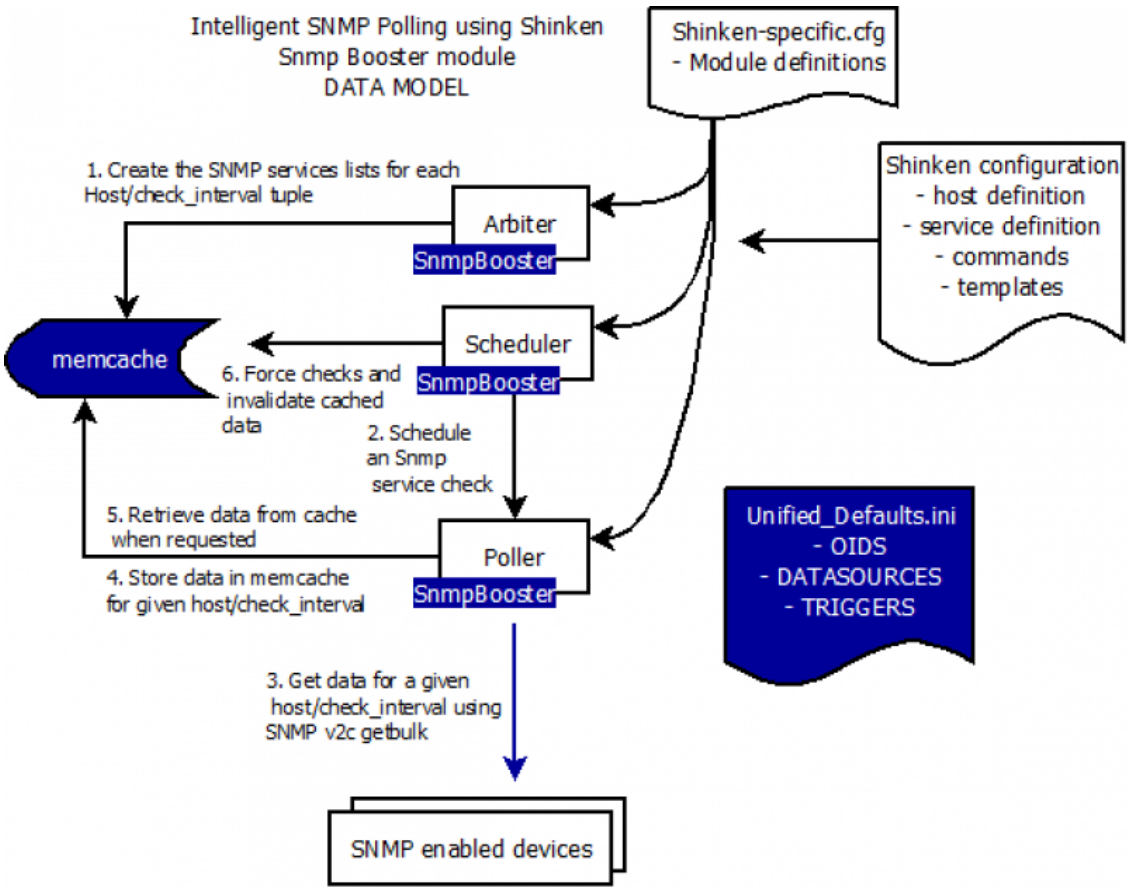

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

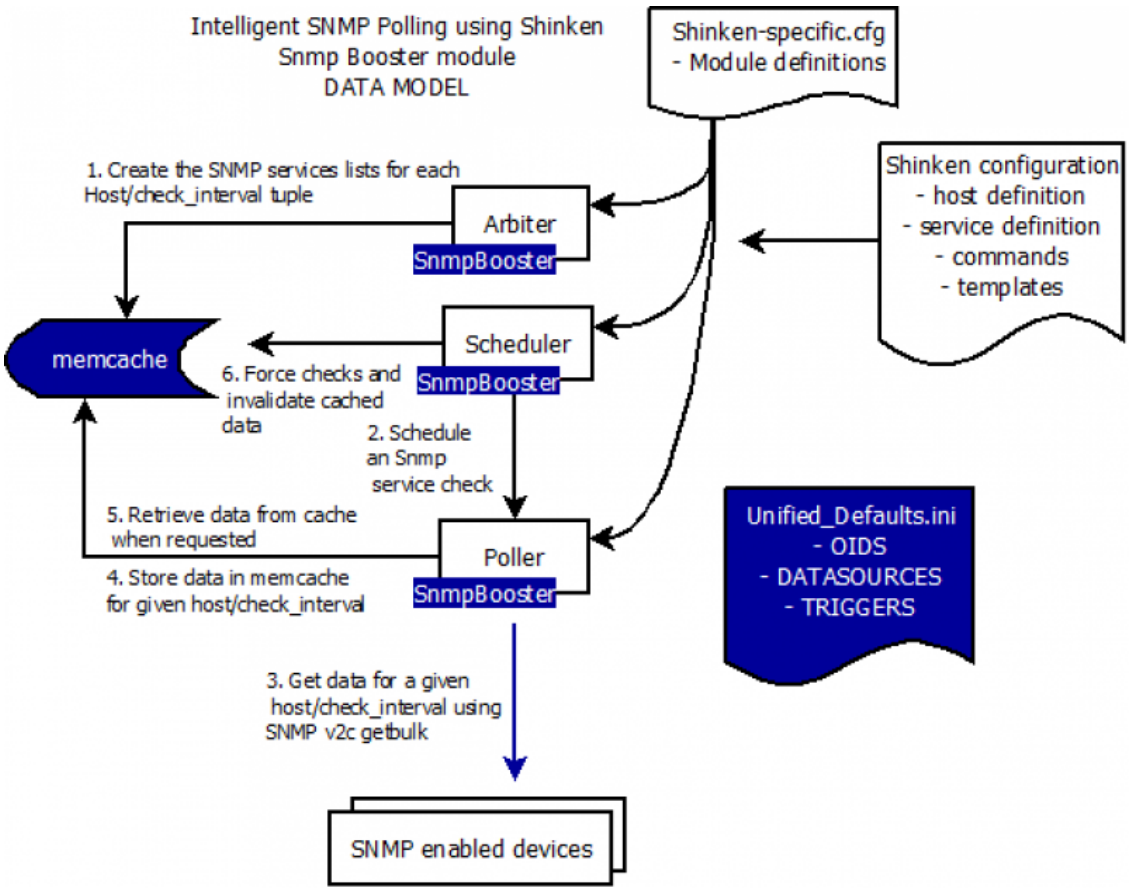

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'**_):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

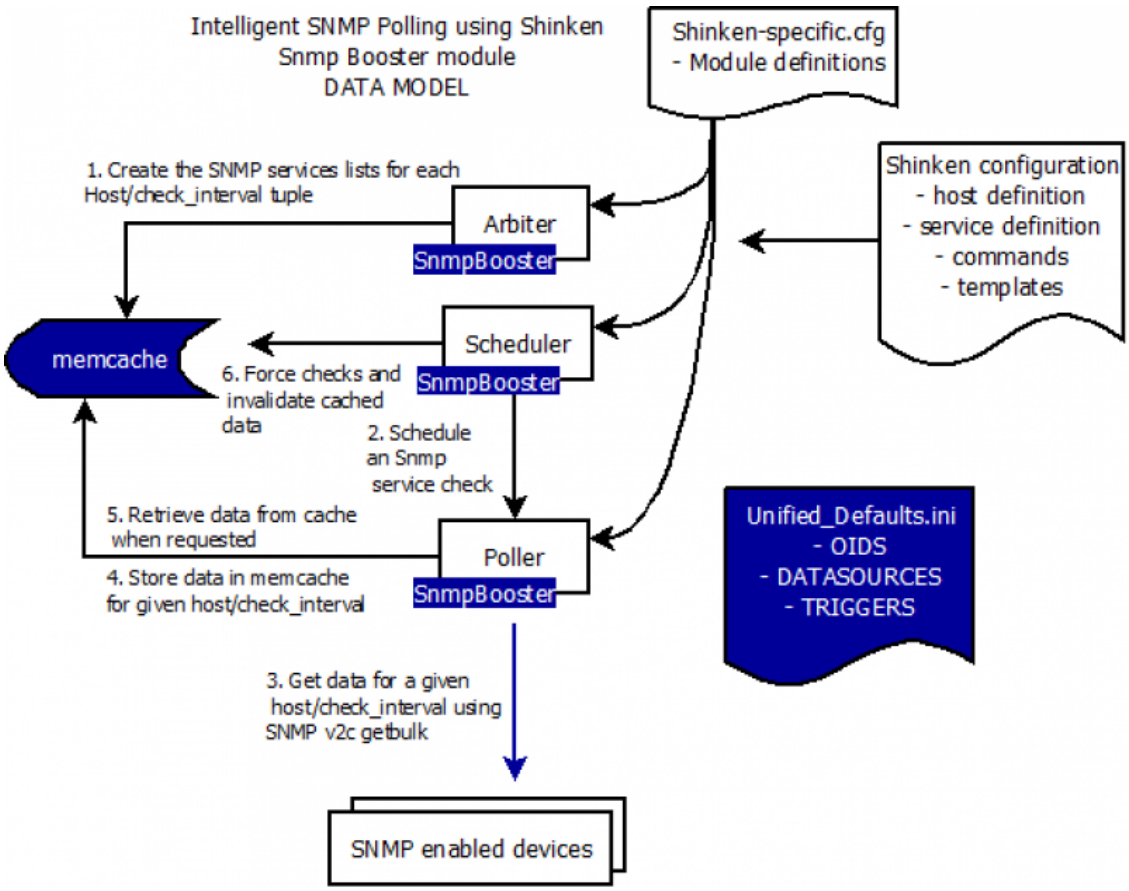

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

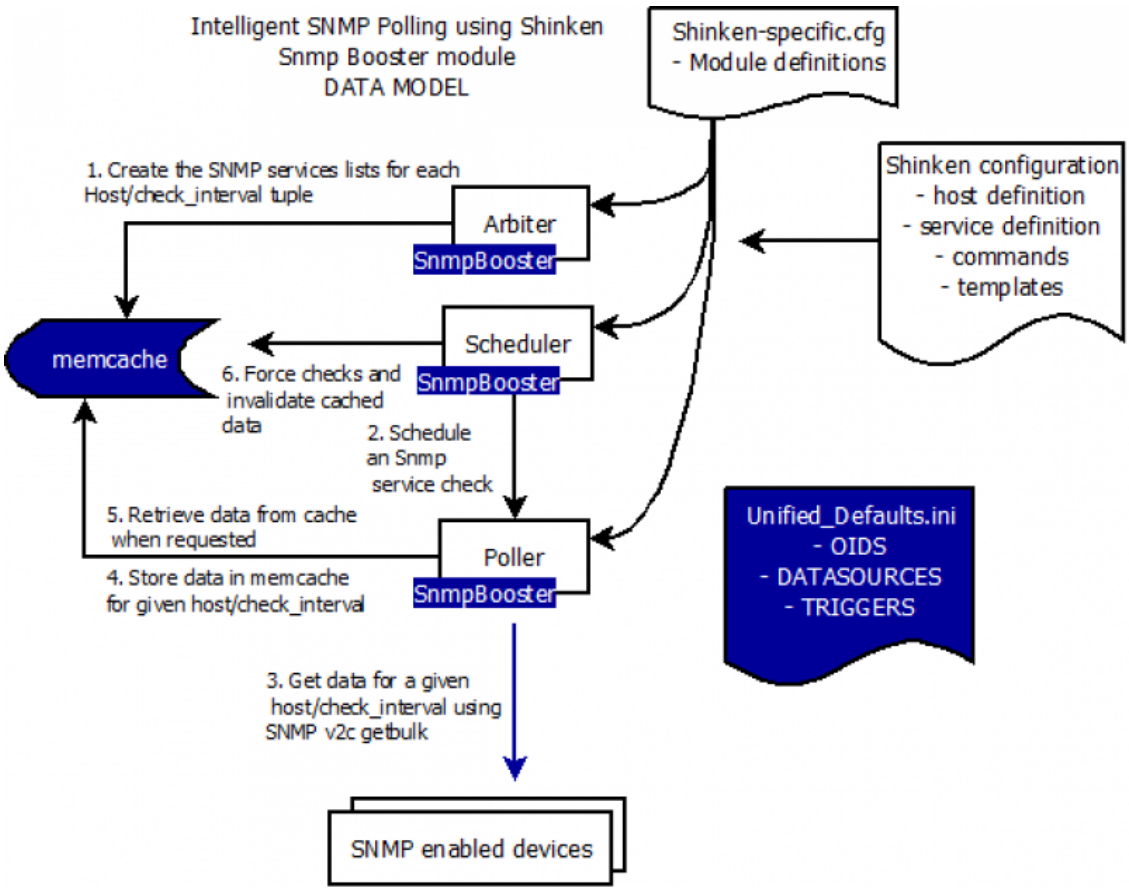

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

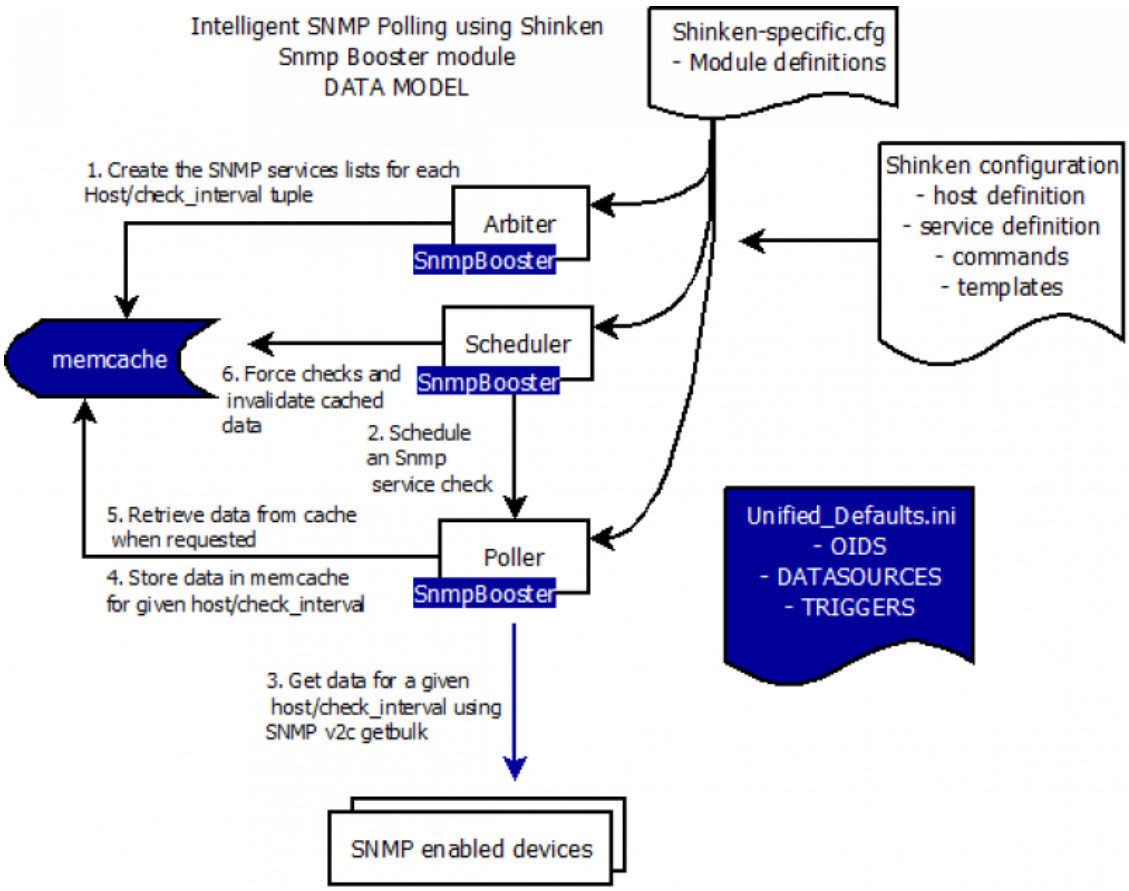

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

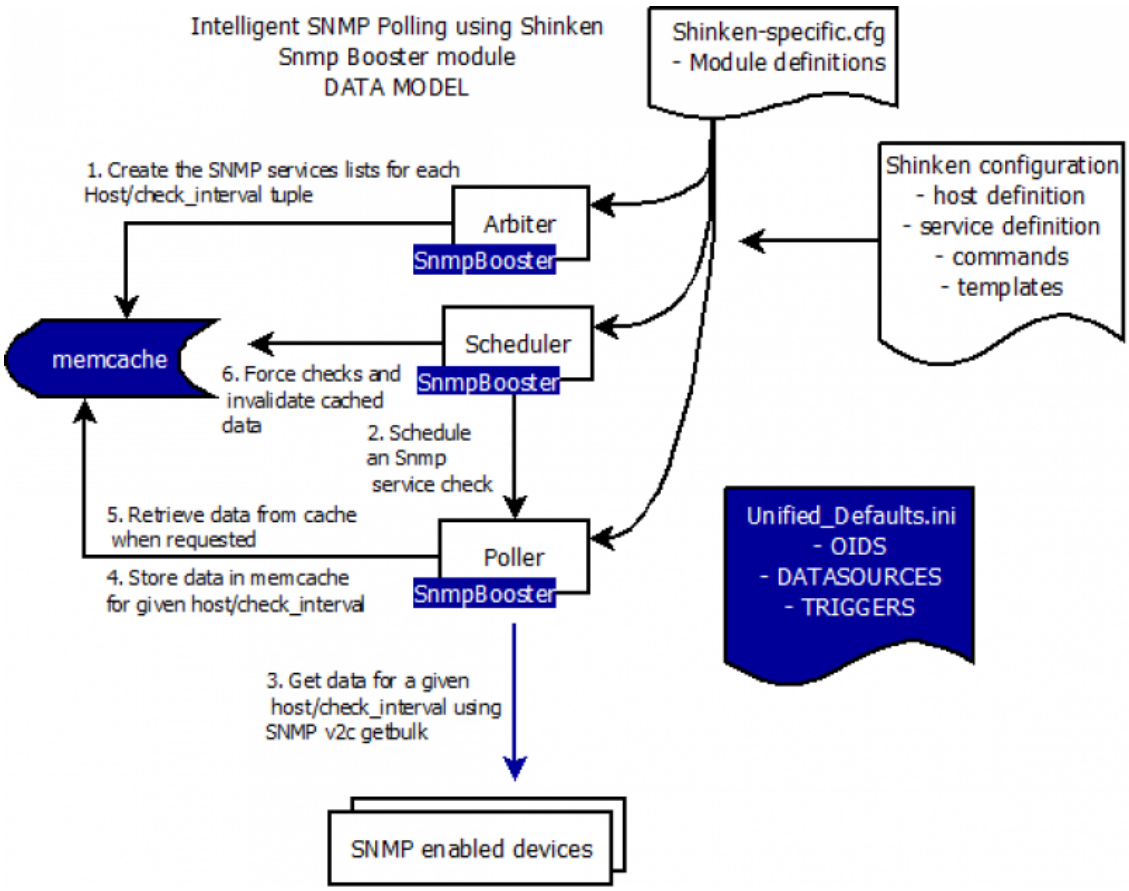

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

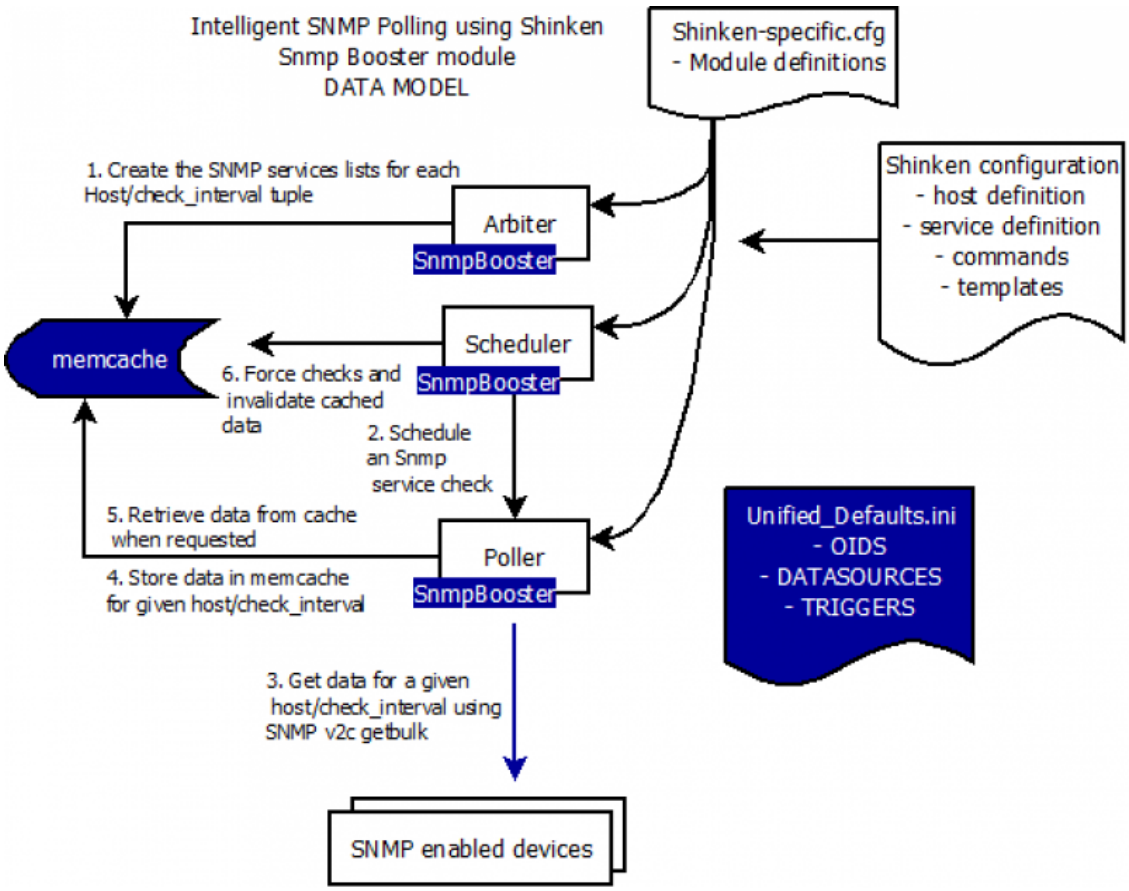

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

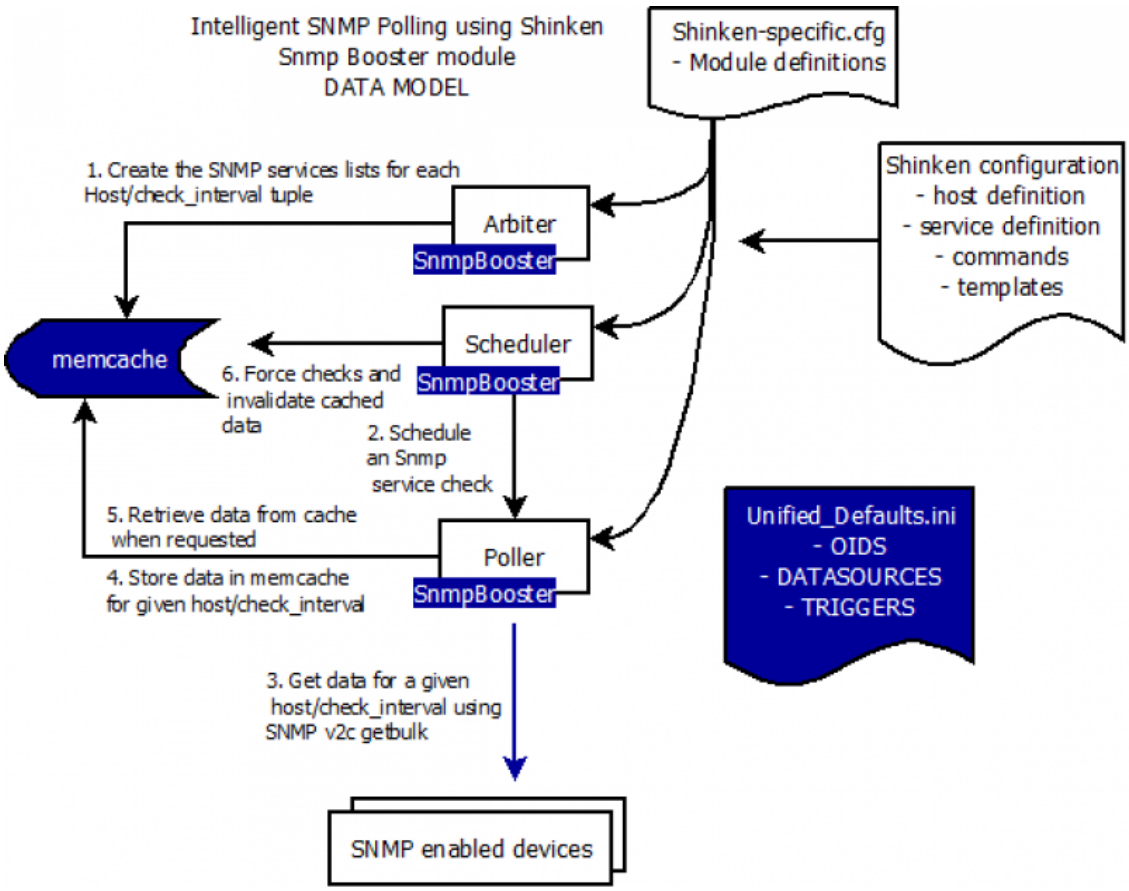

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

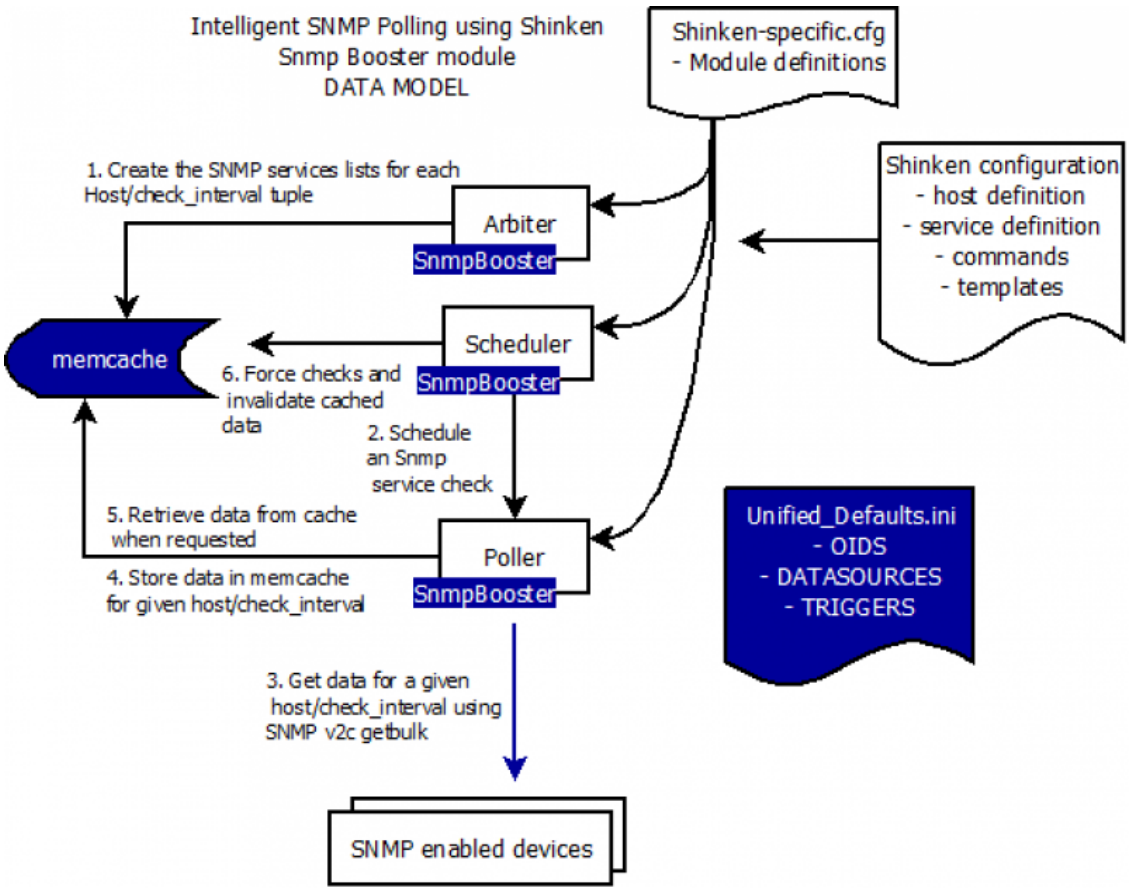

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

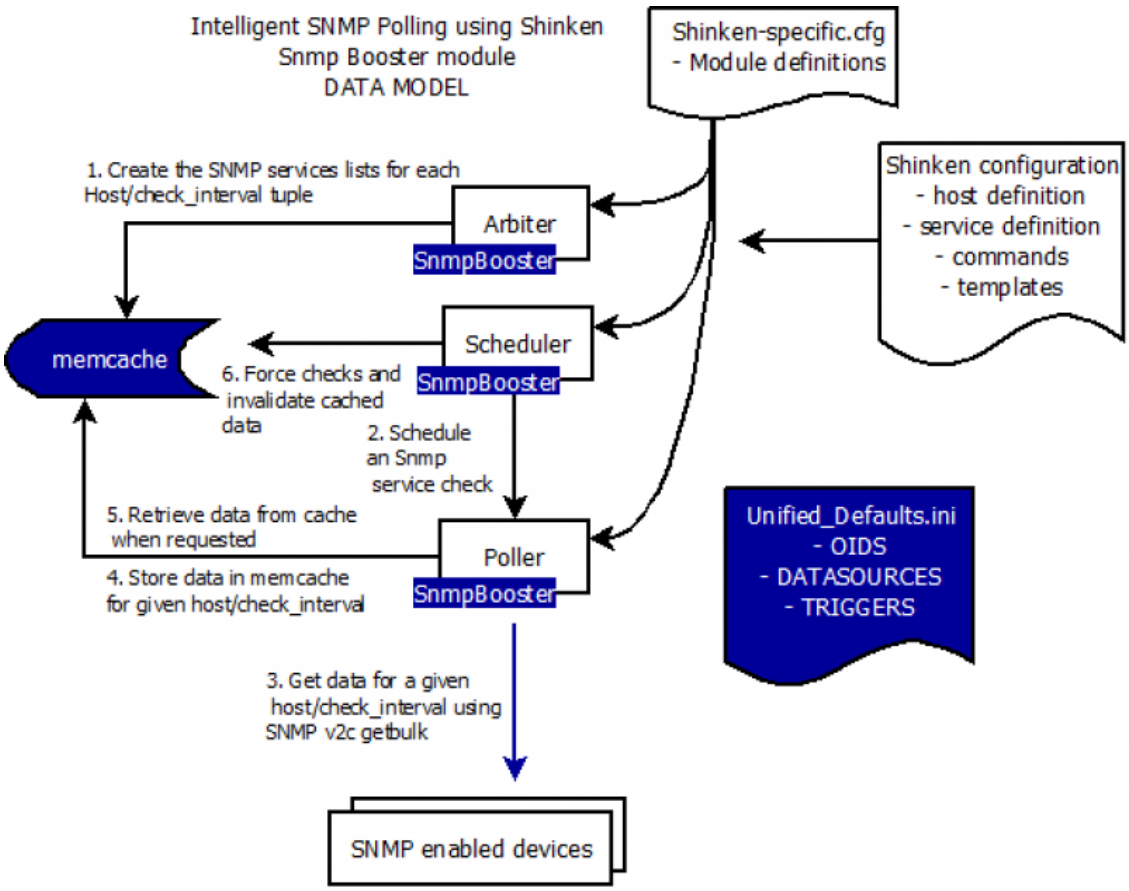

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

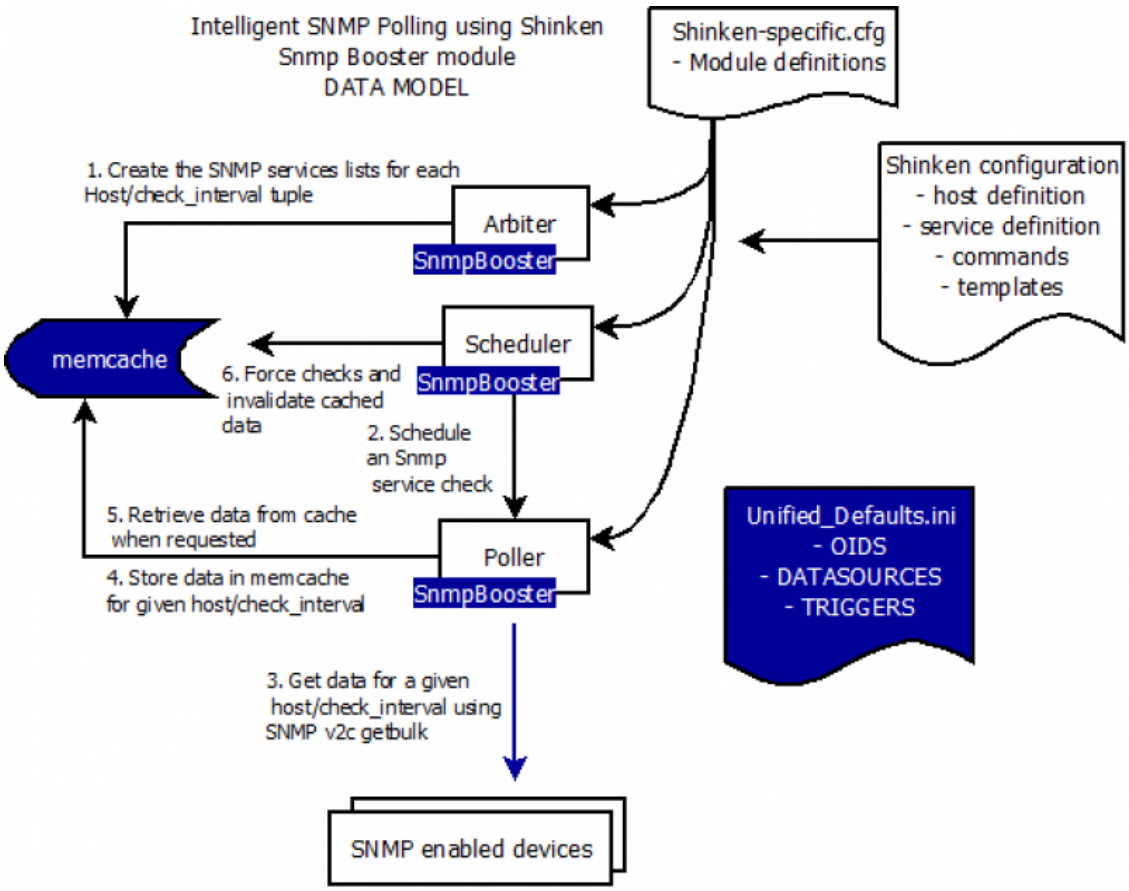

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

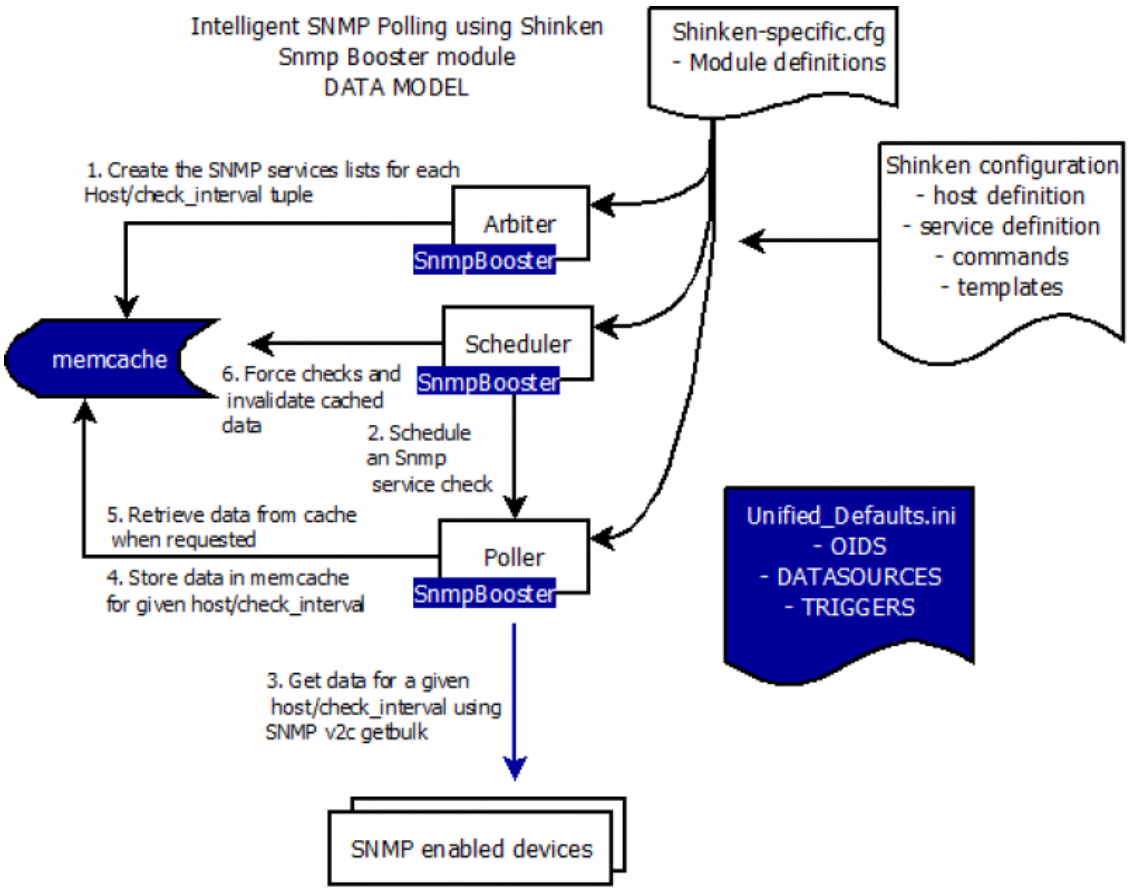

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

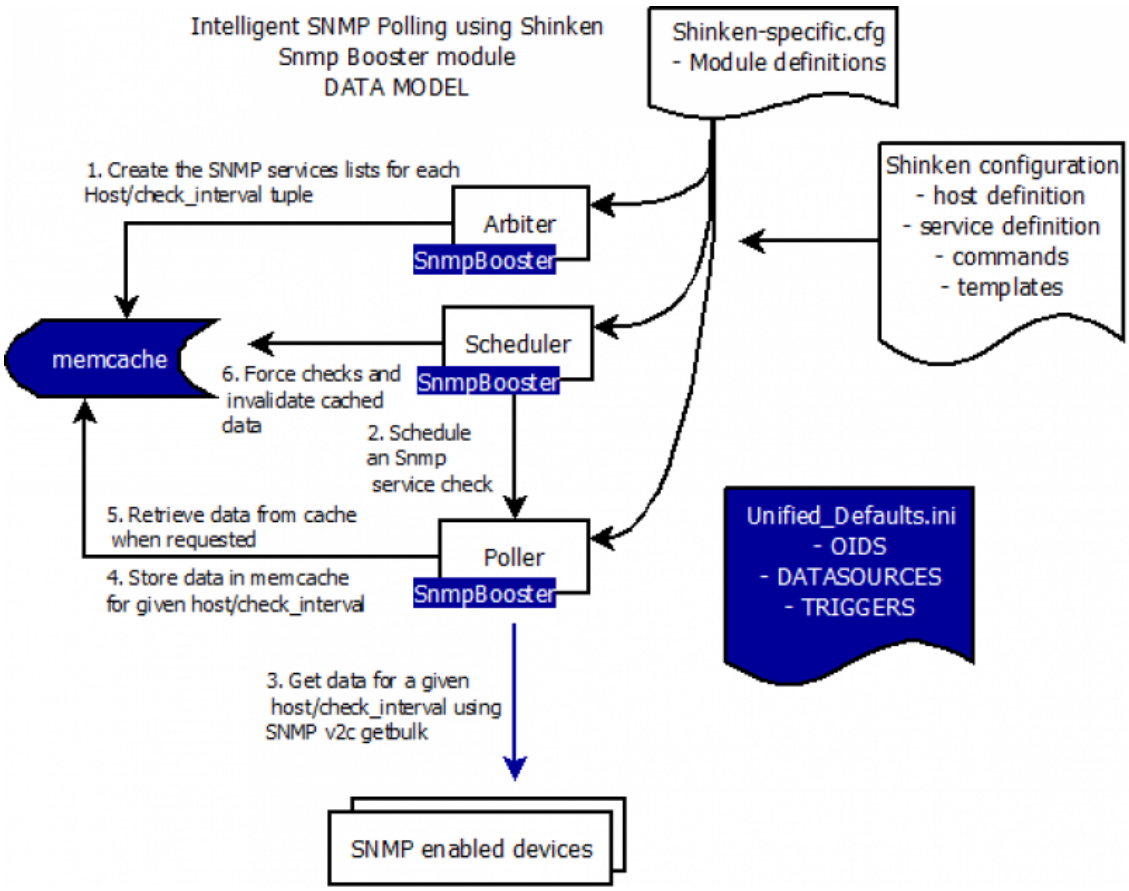

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

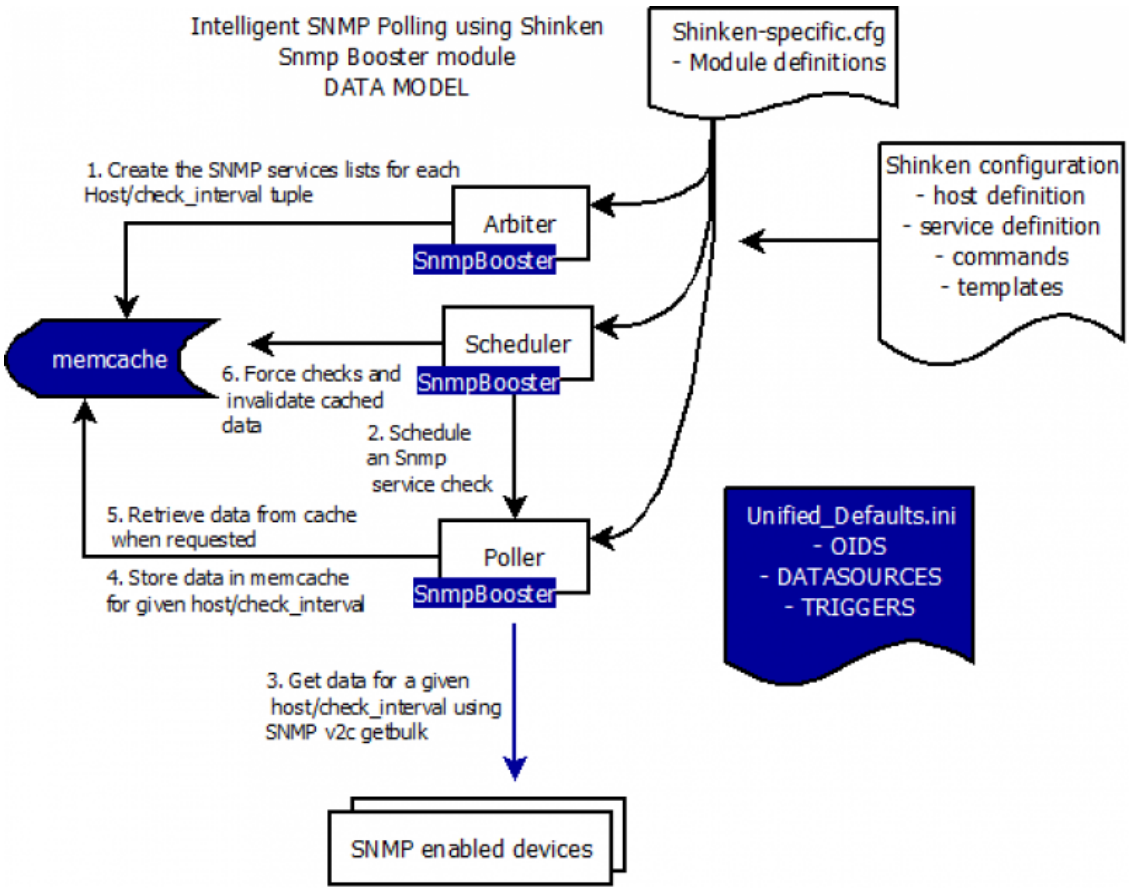

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

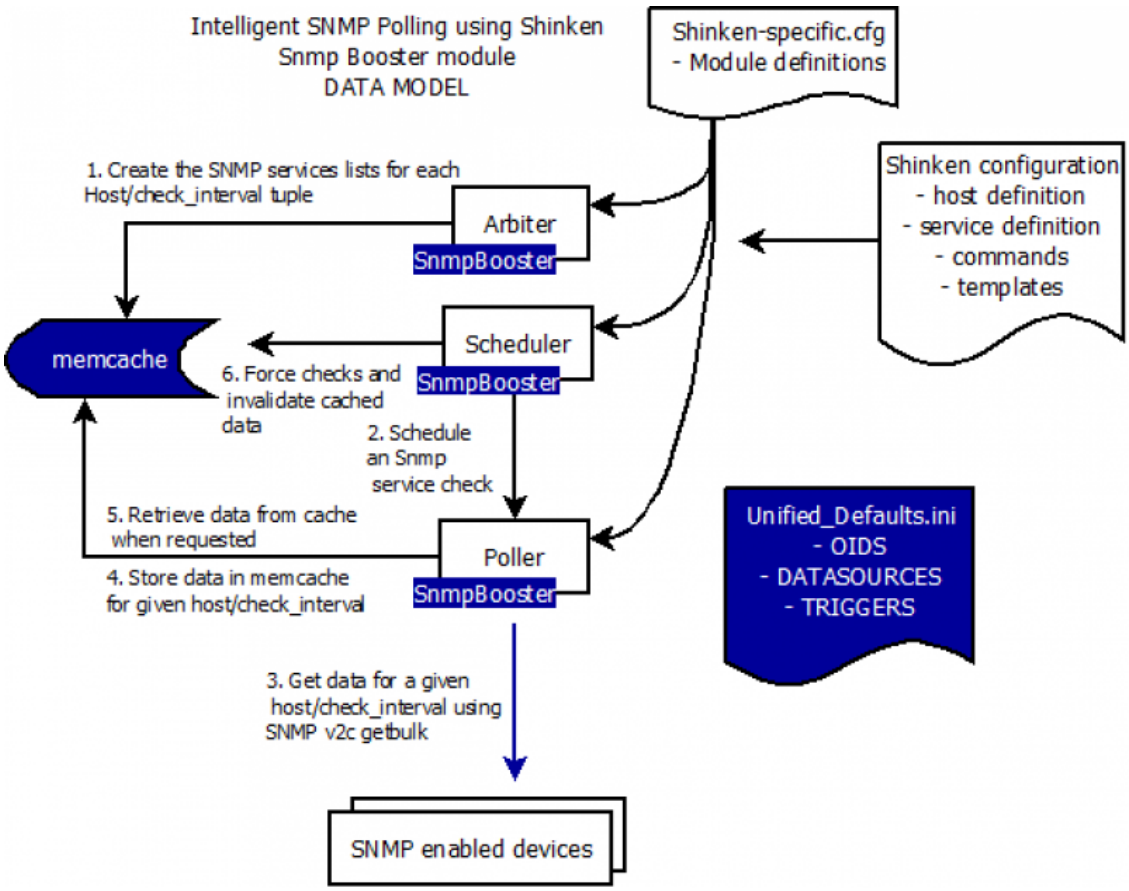

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

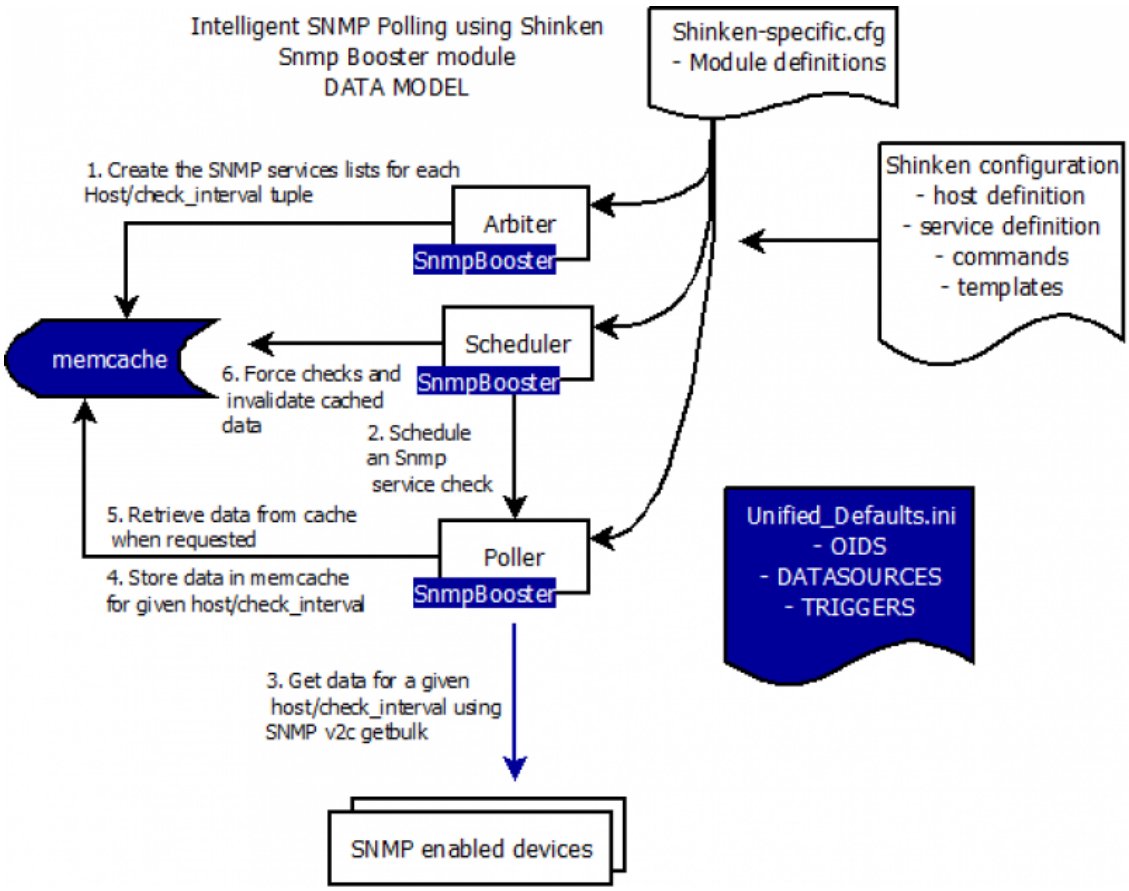

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

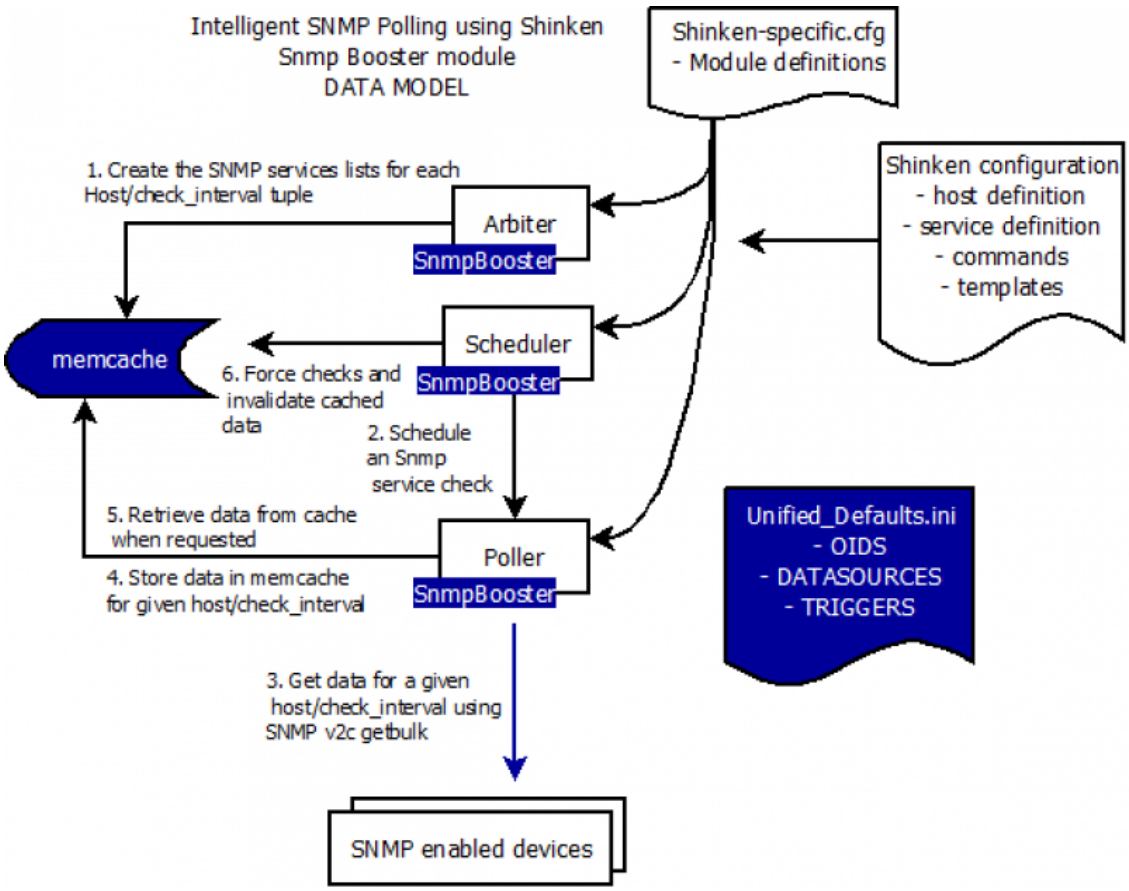

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

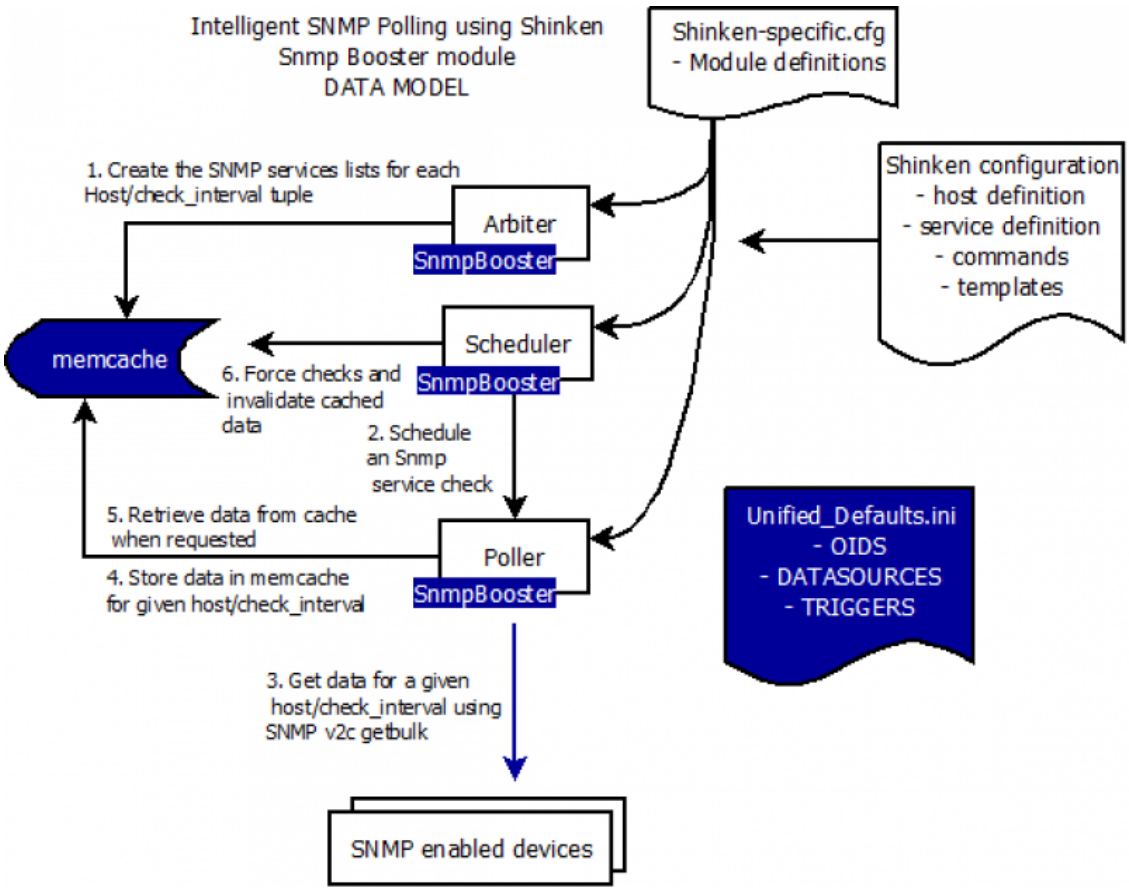

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

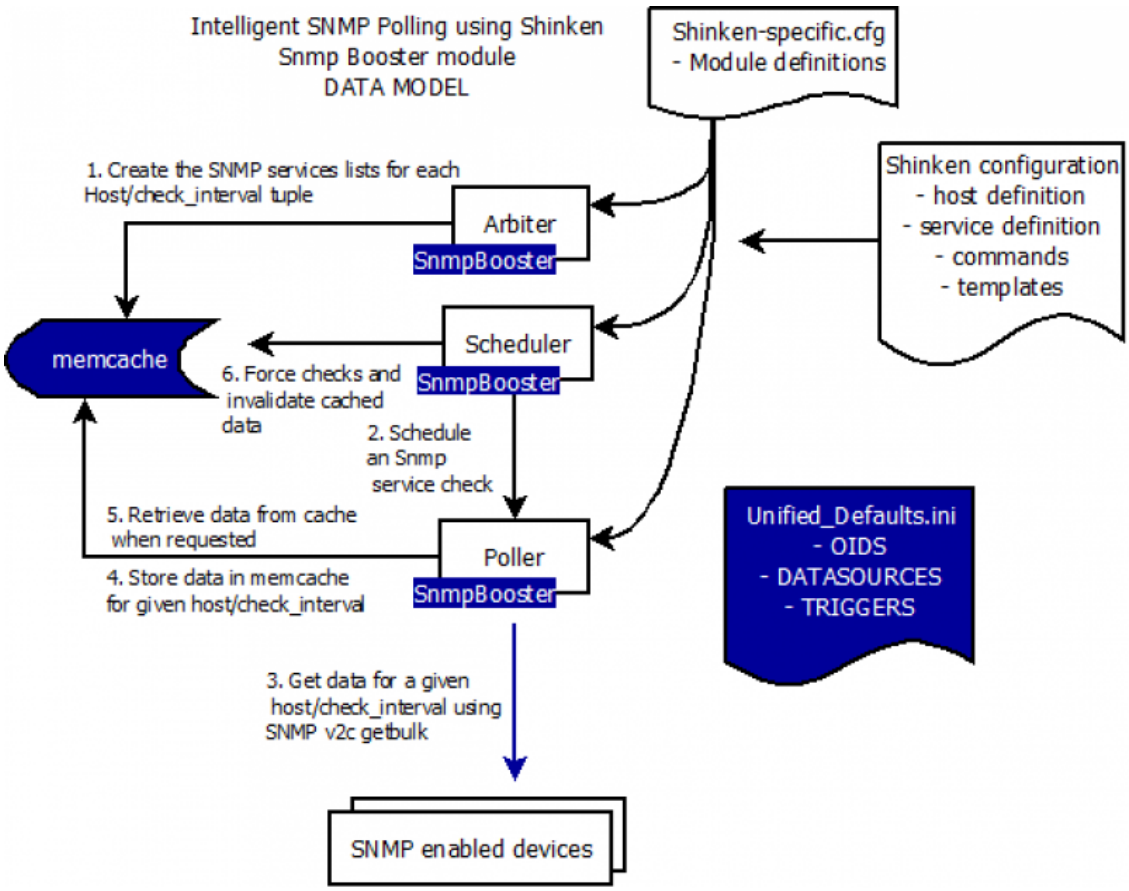

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

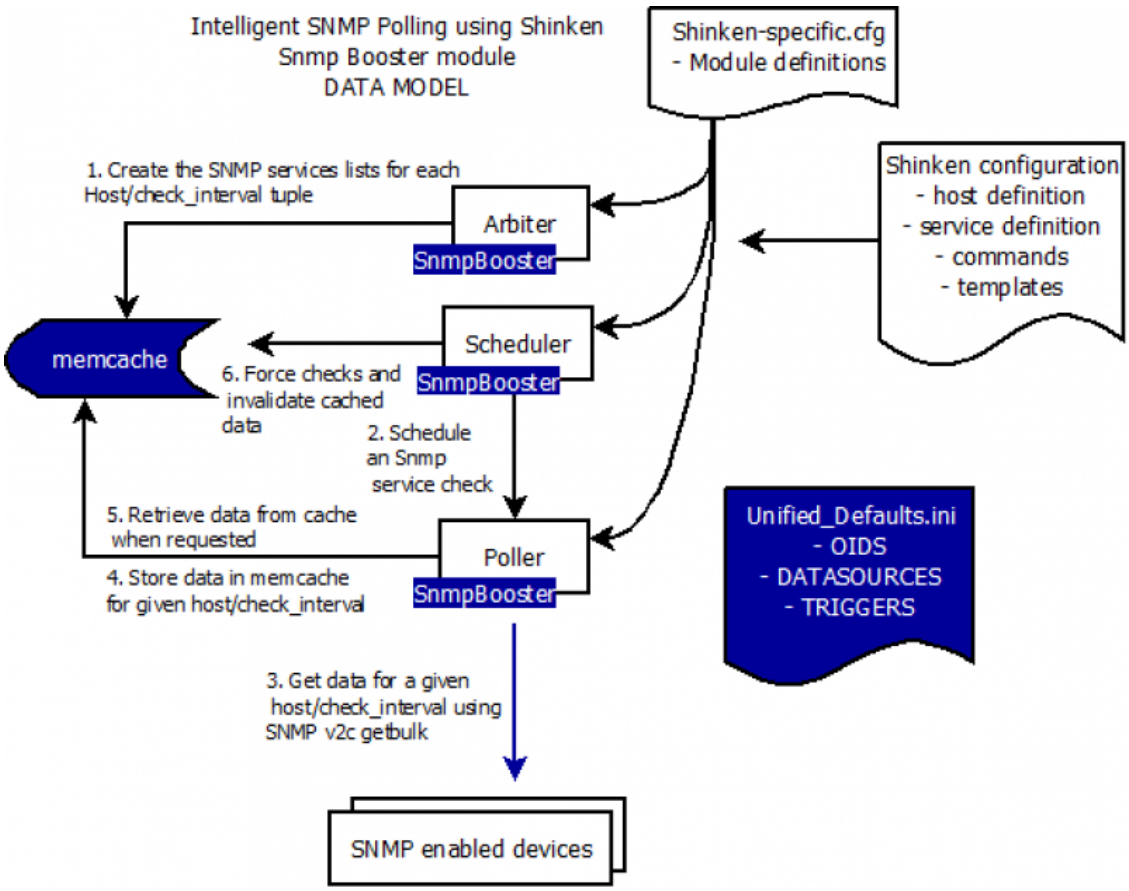

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

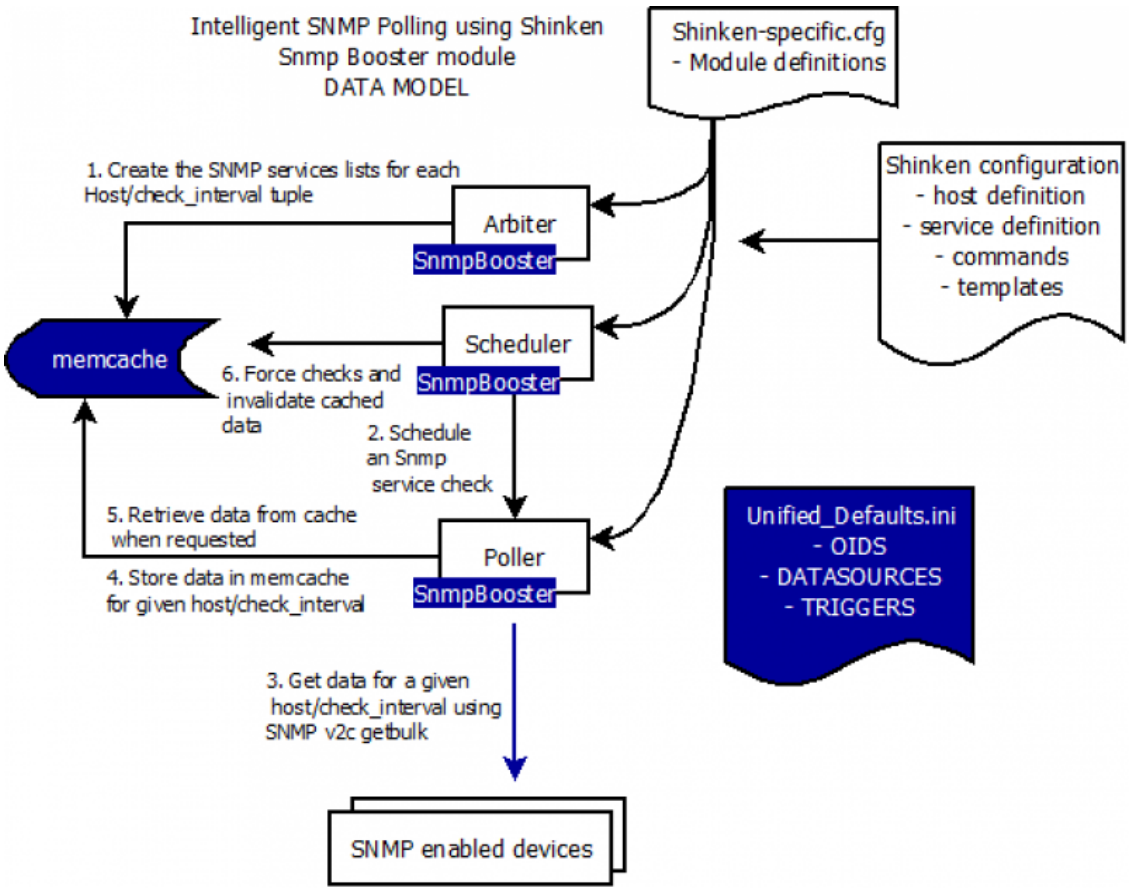

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

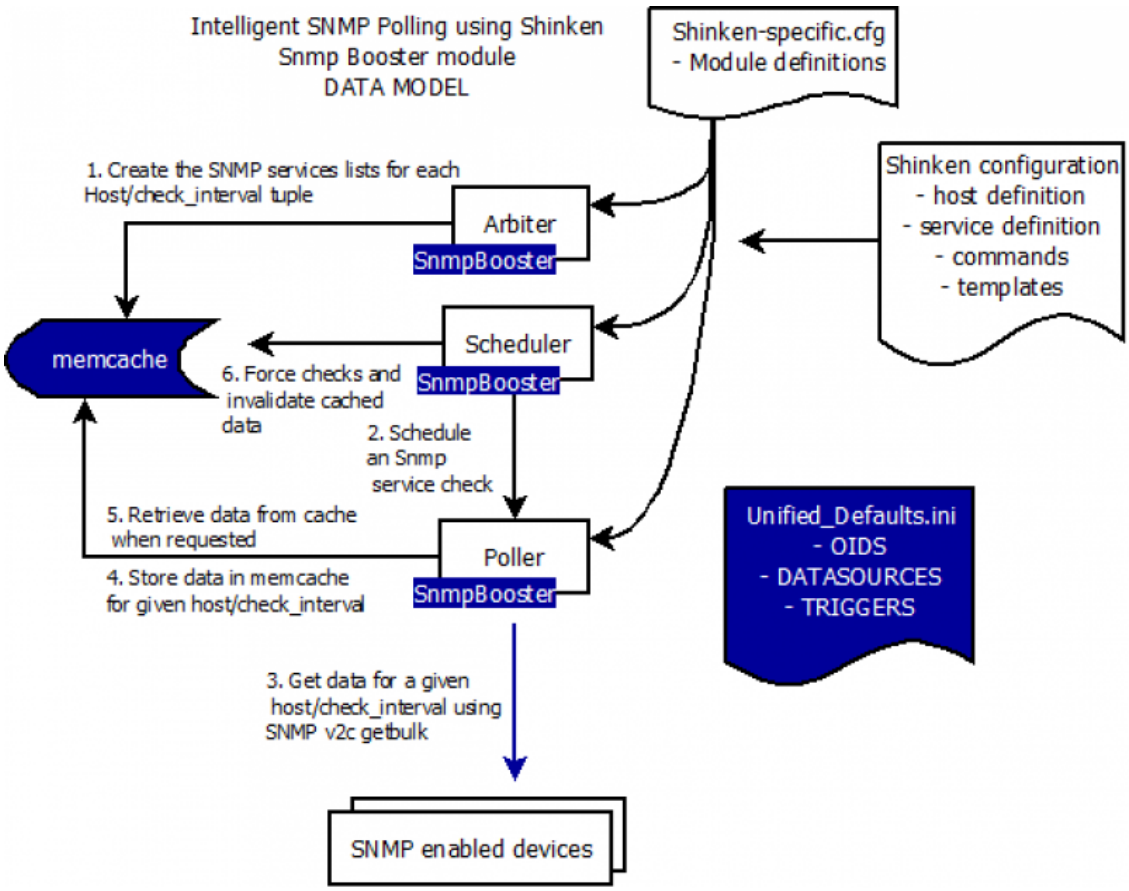

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

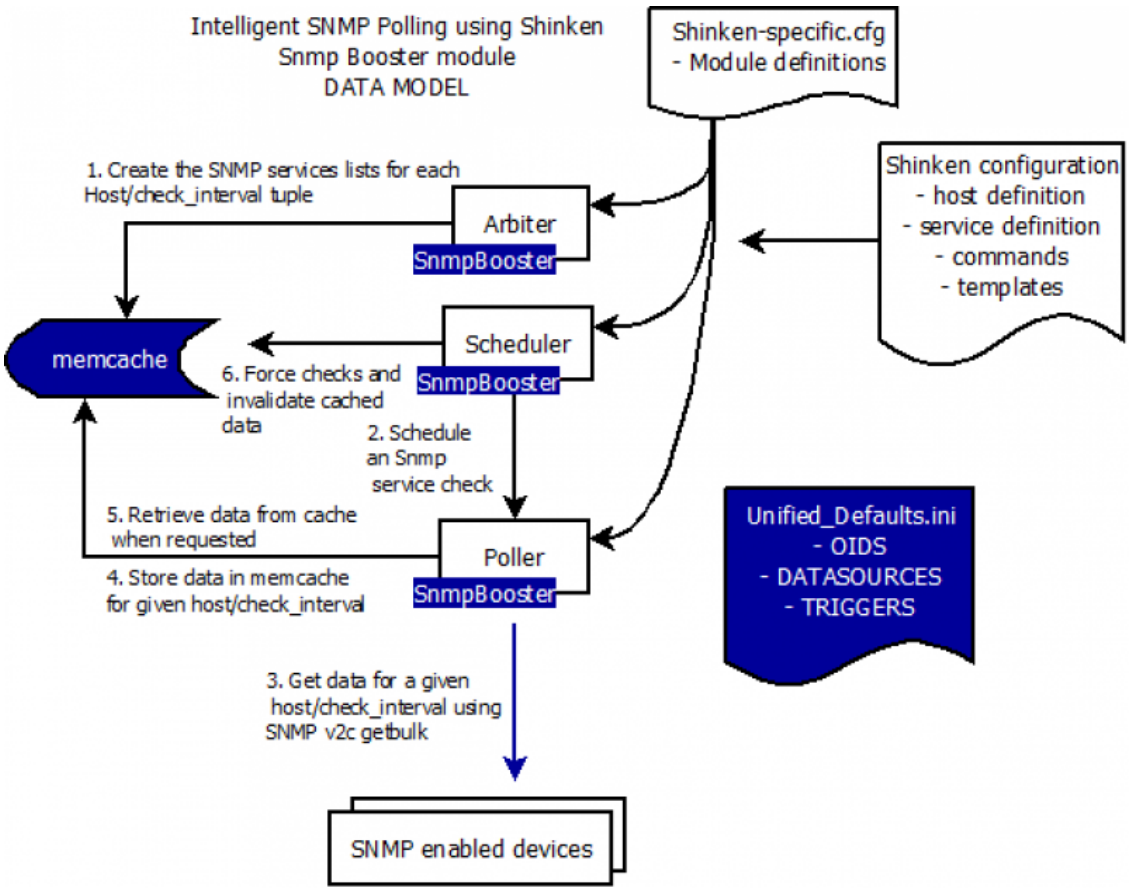

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

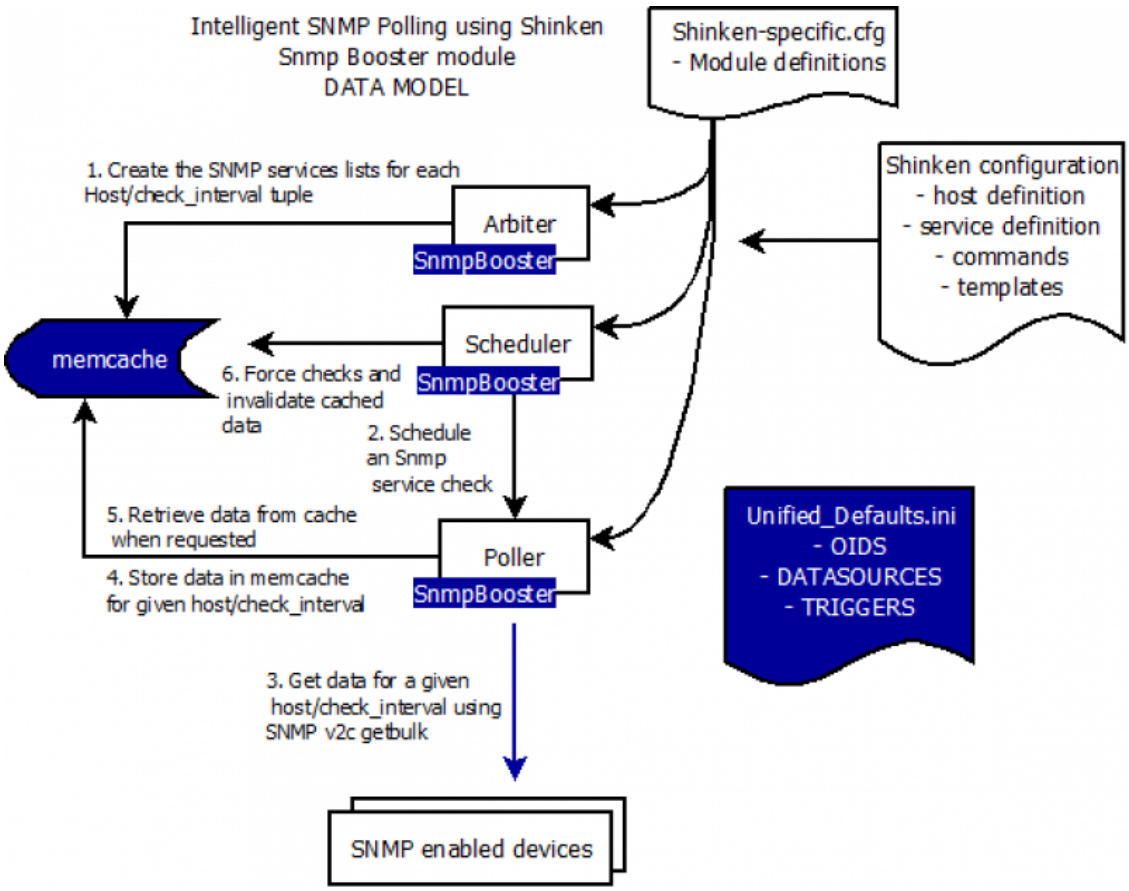

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

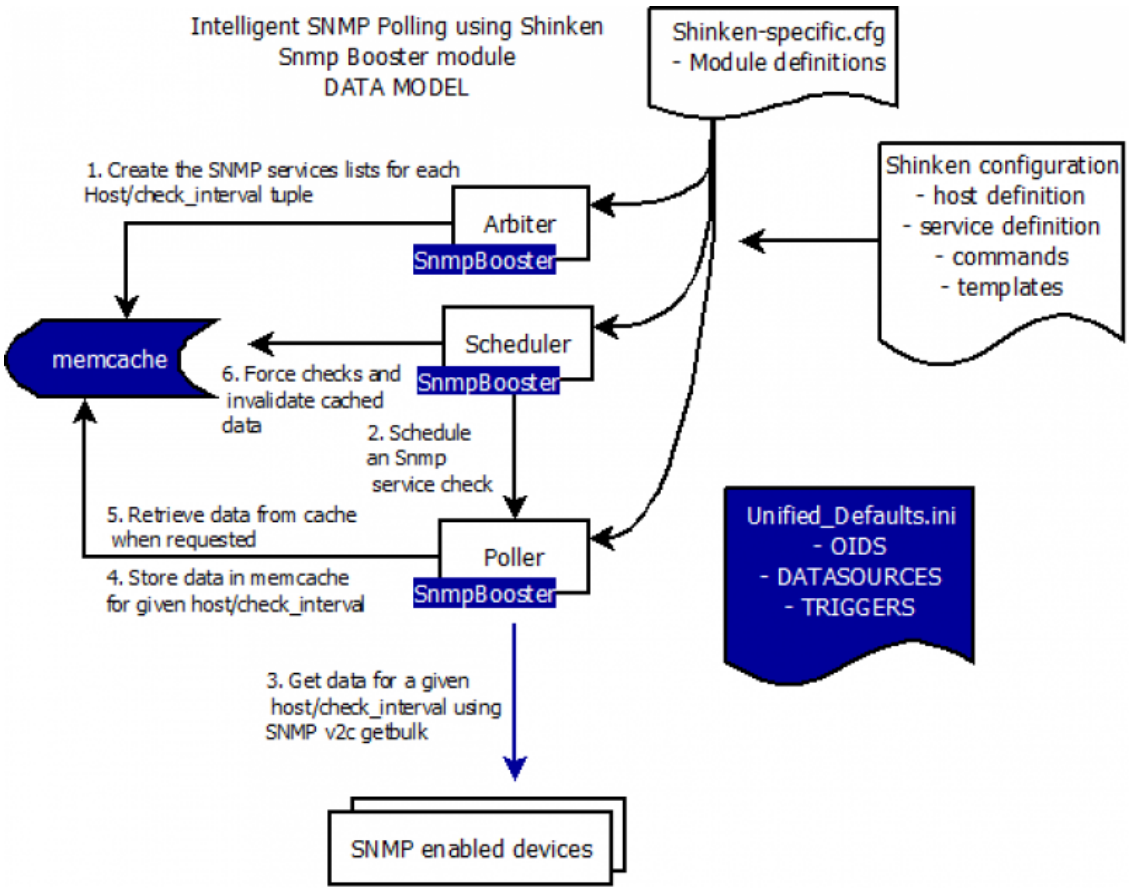

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

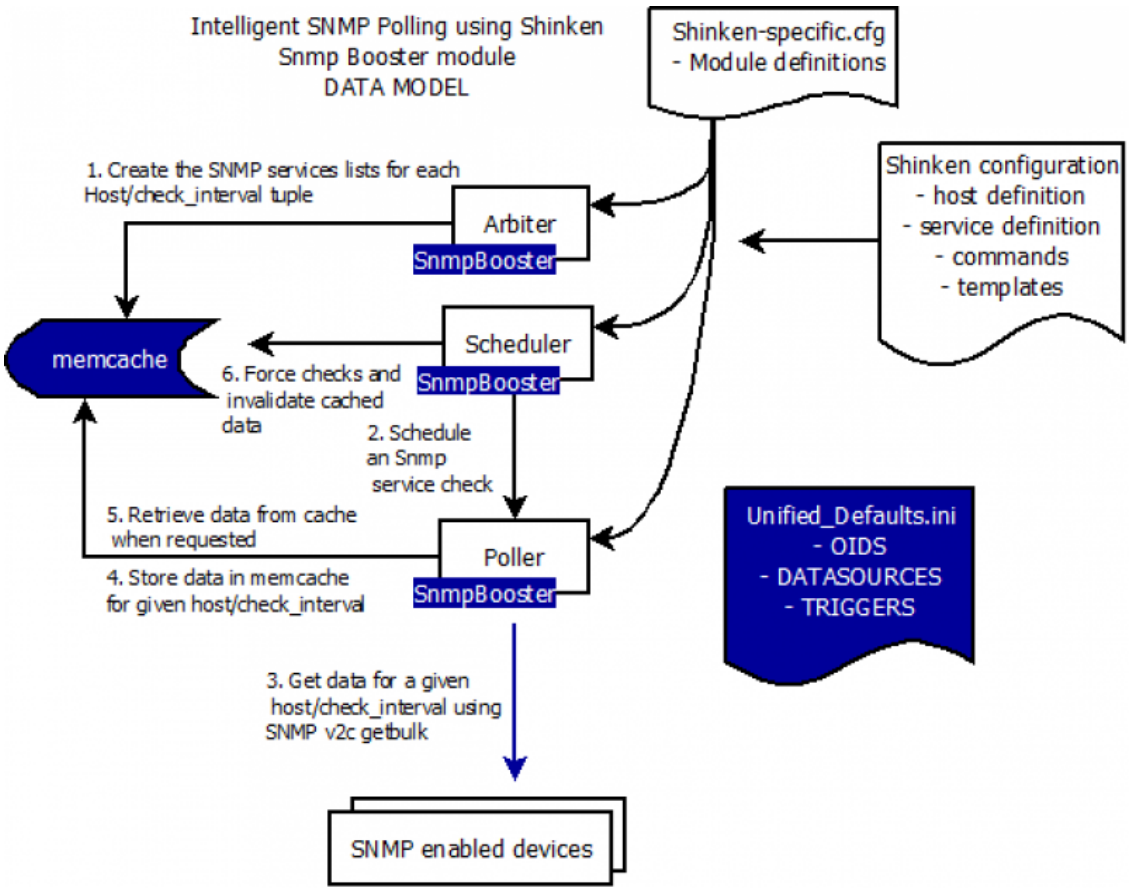

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

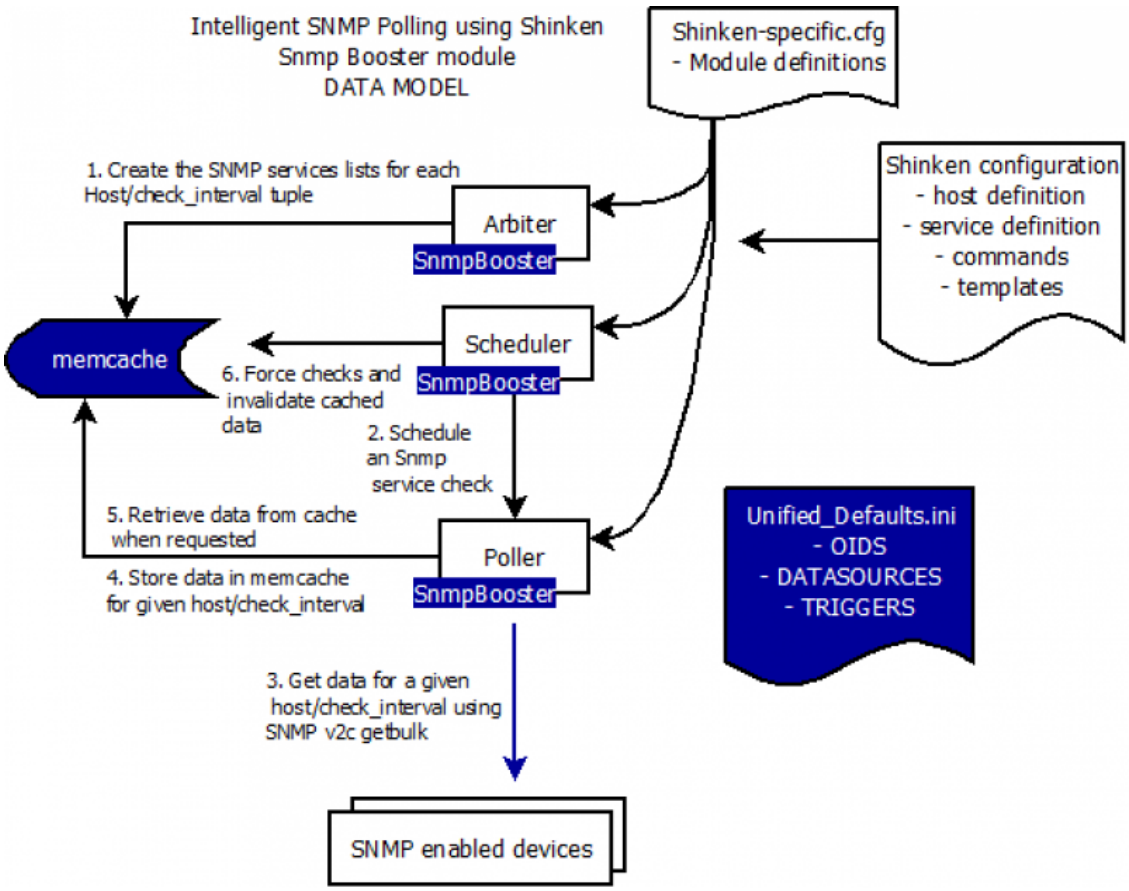

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

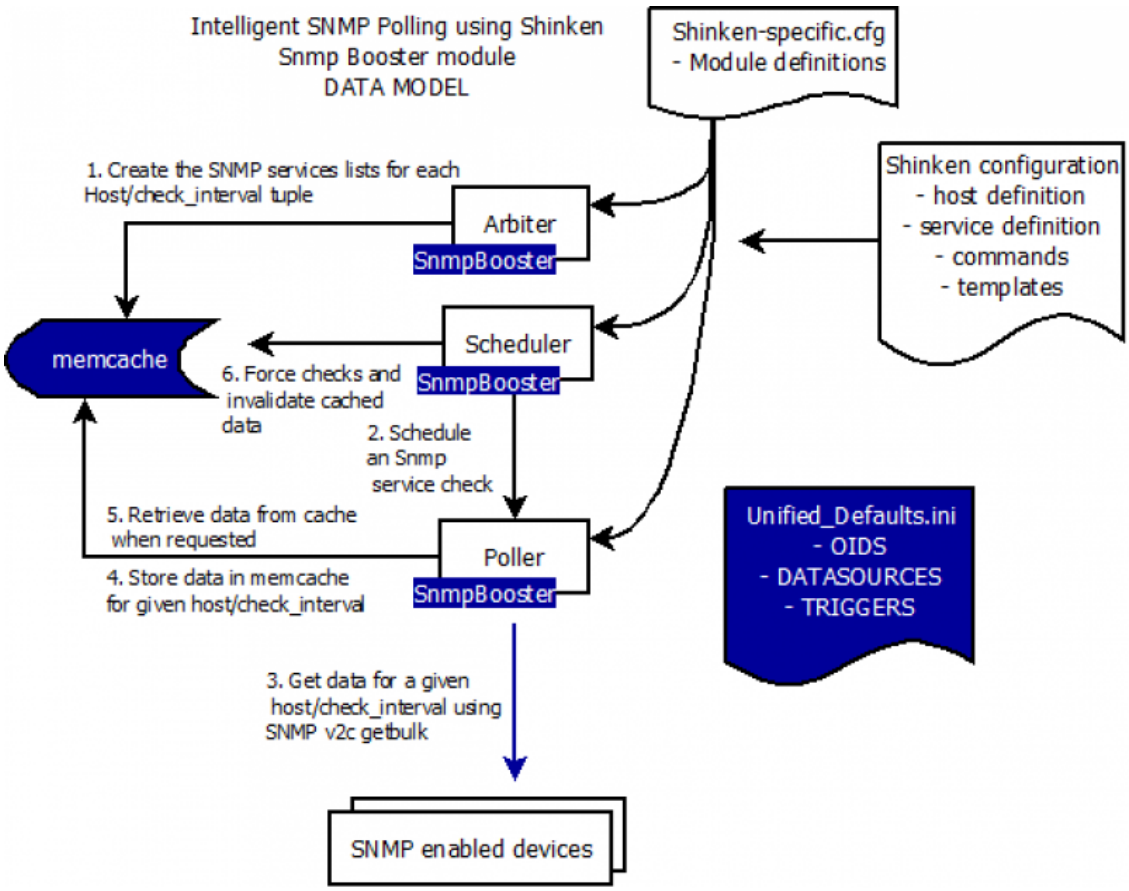

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

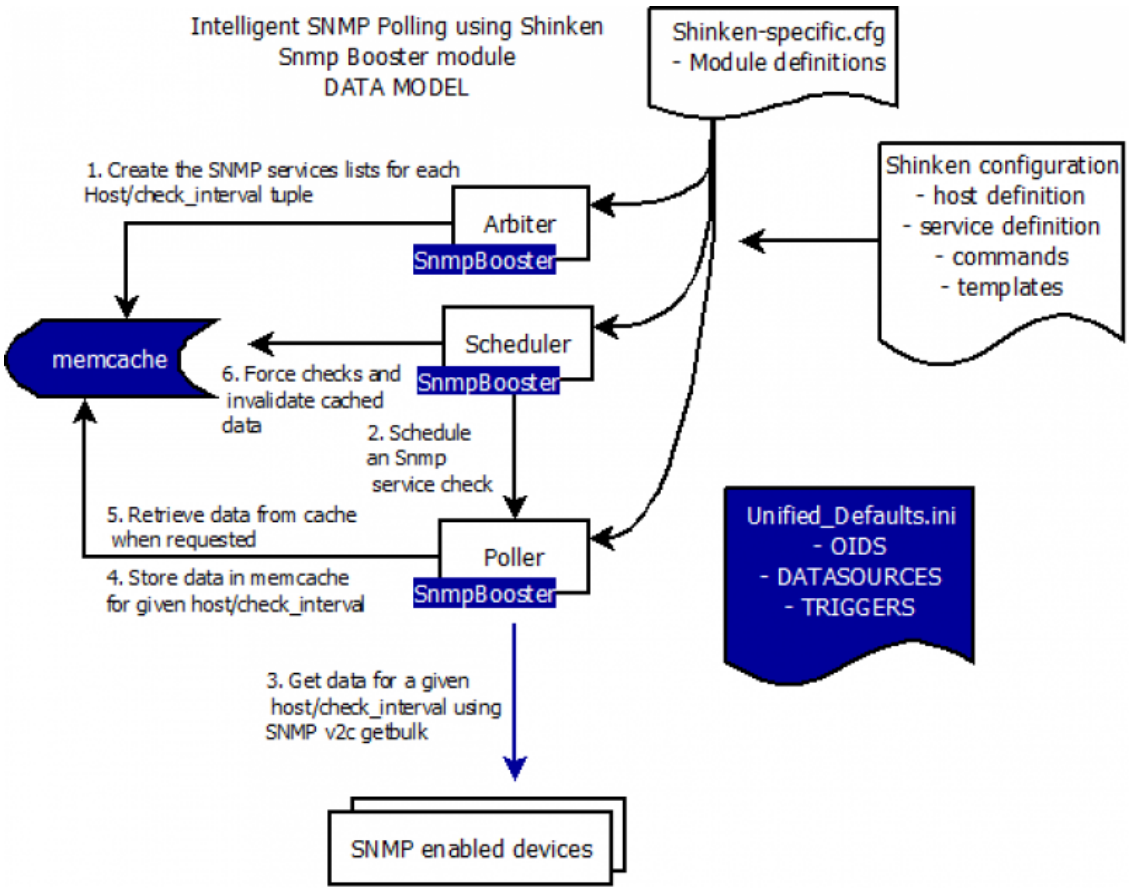

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

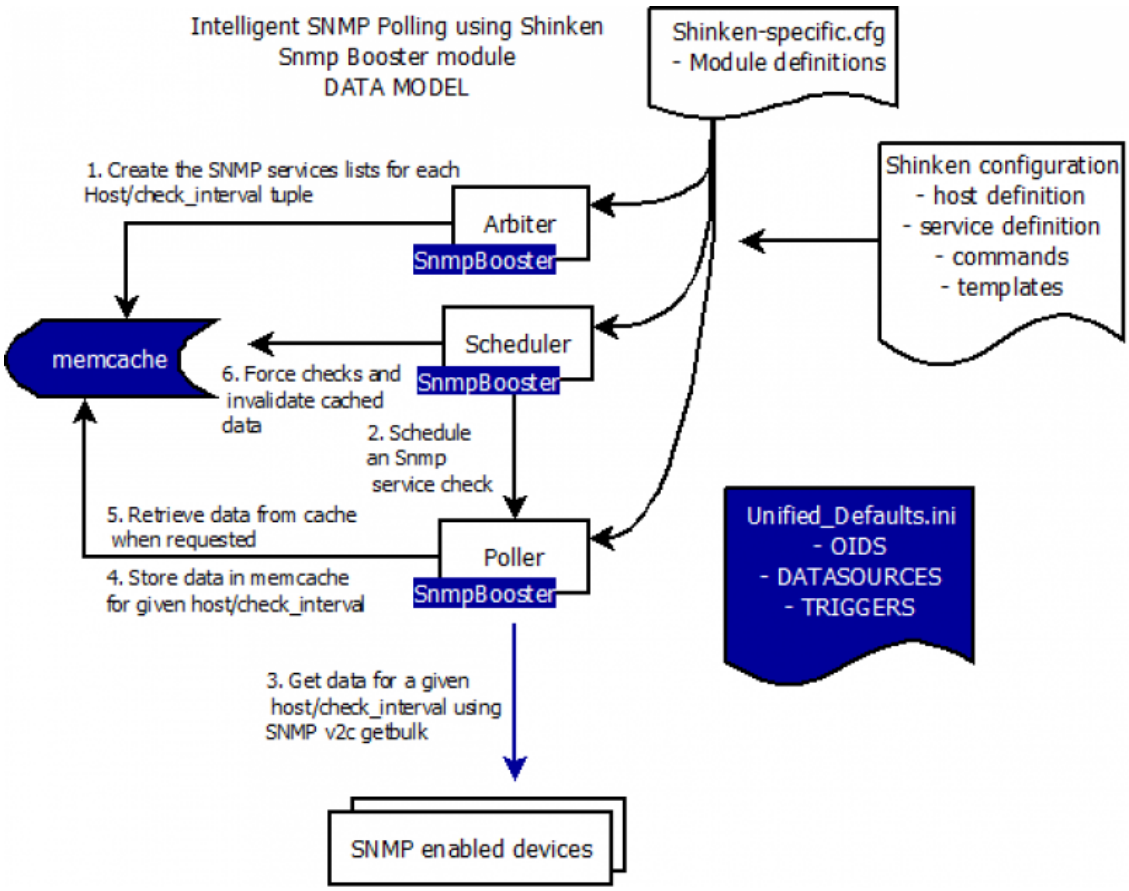

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

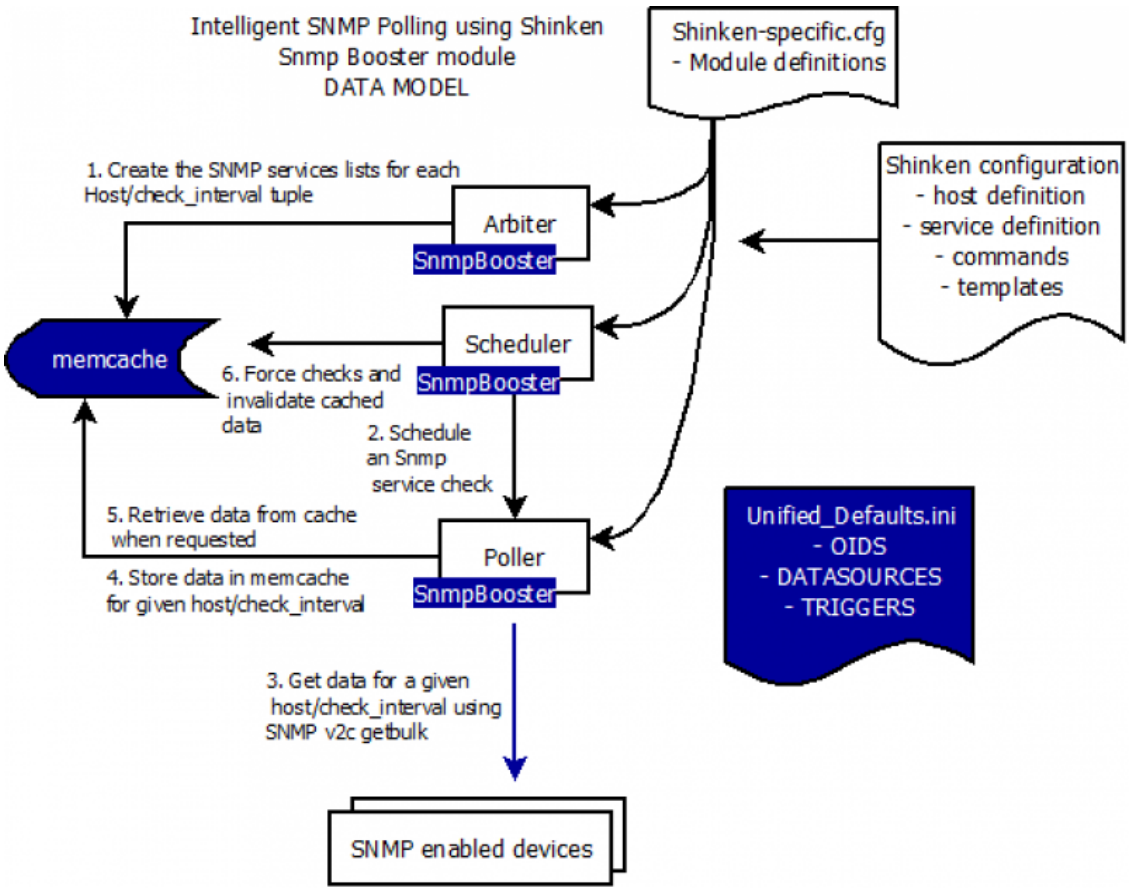

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

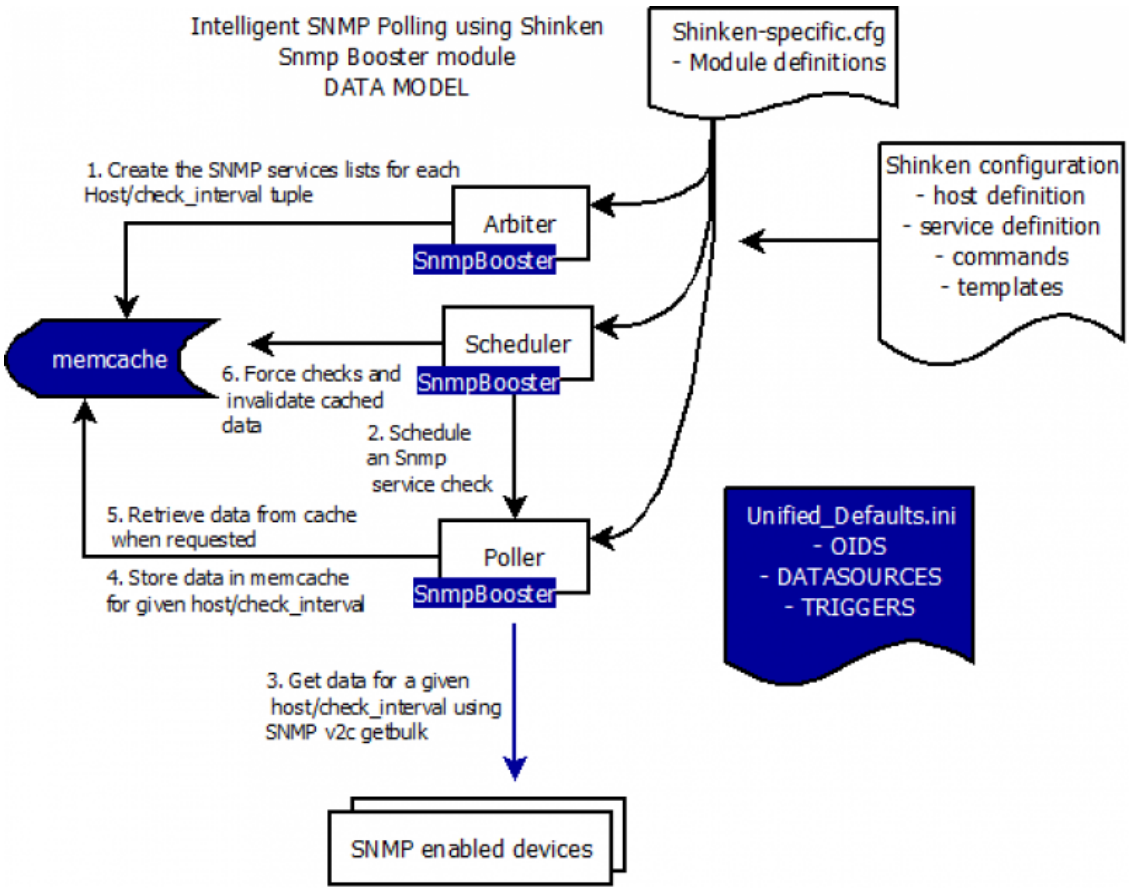

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

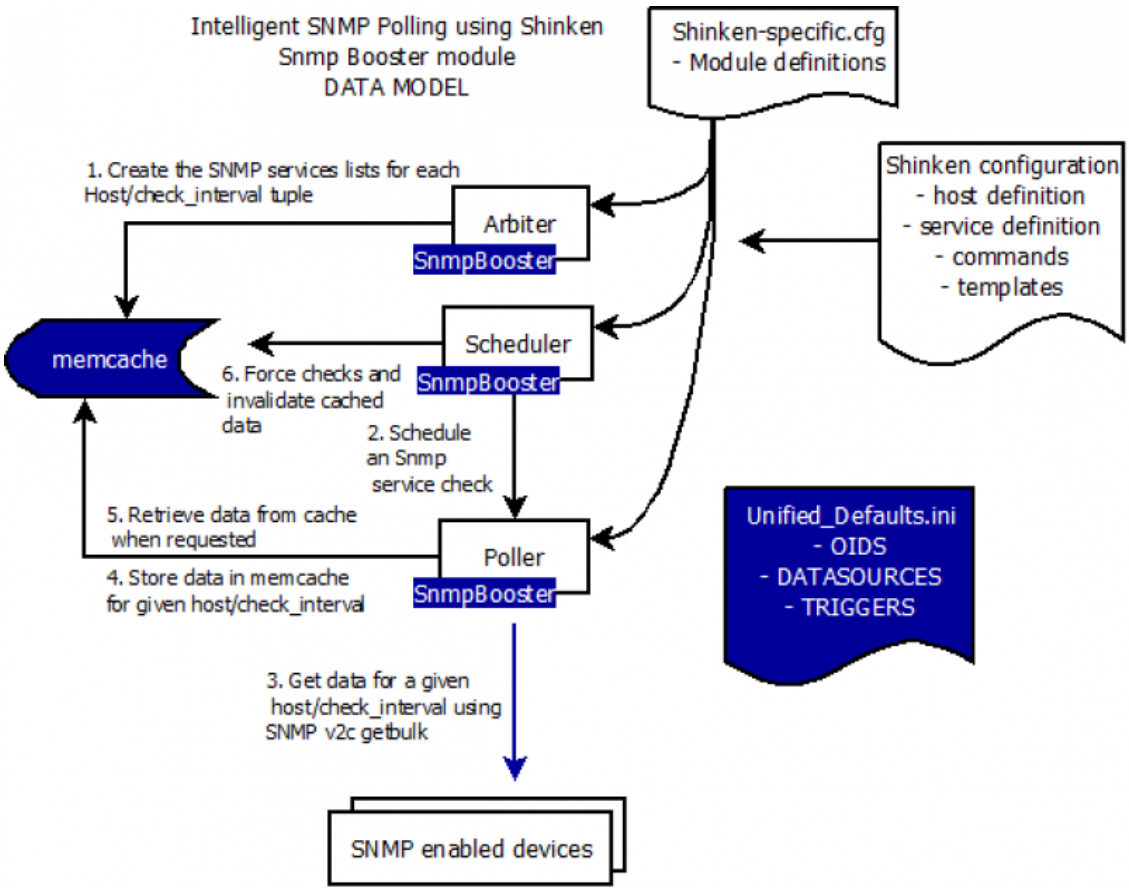

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

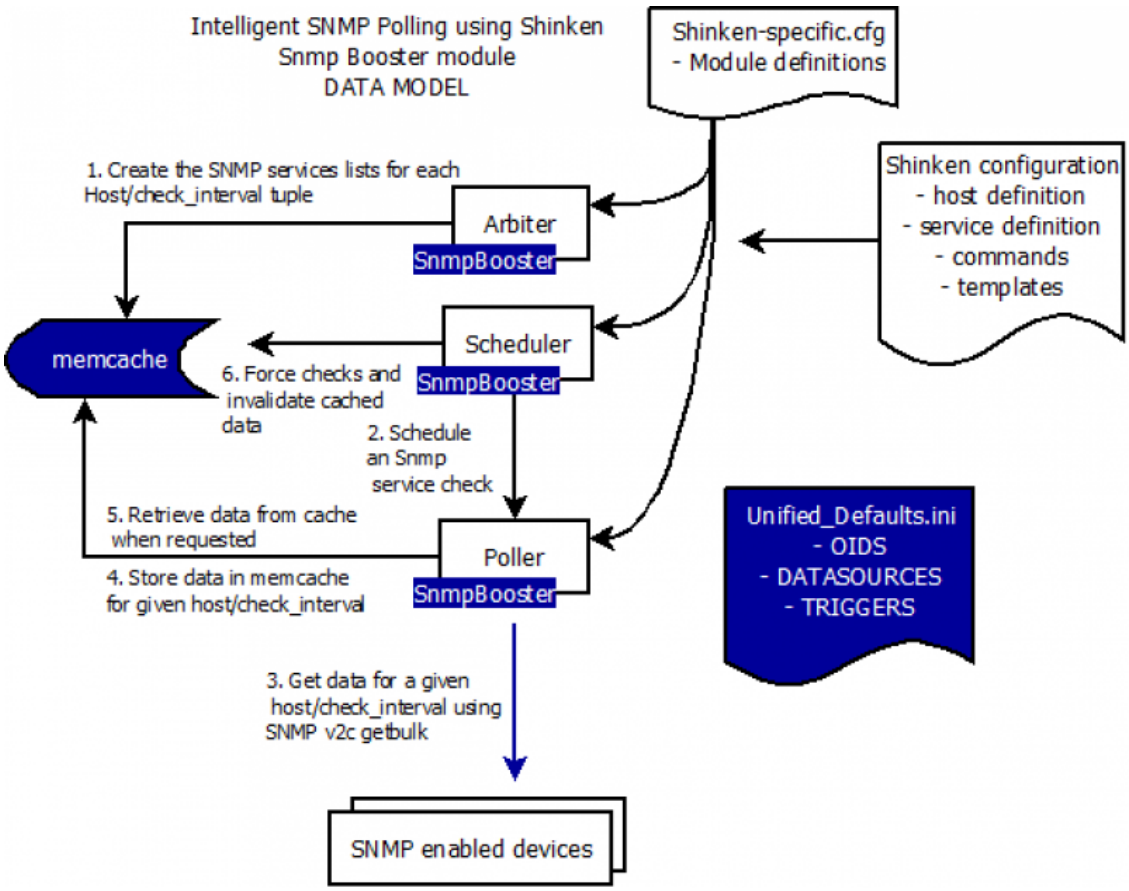

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

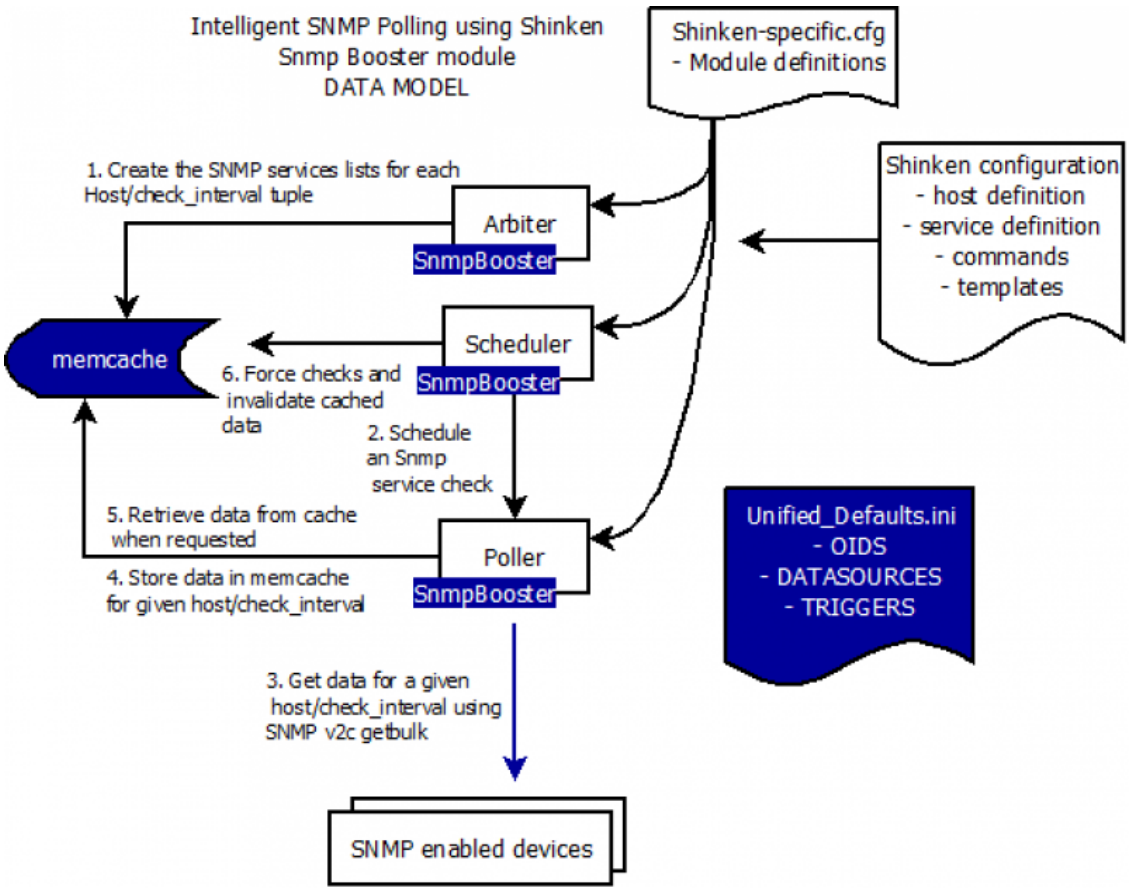

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

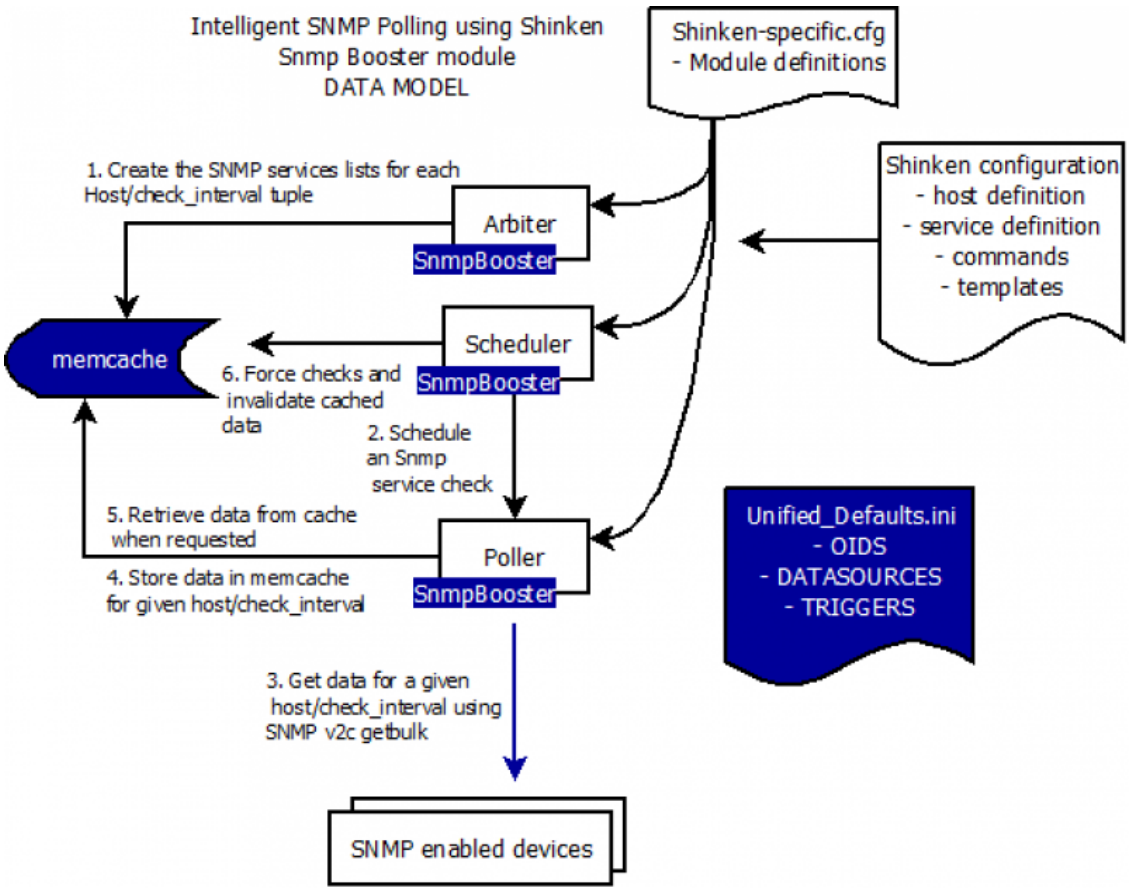

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

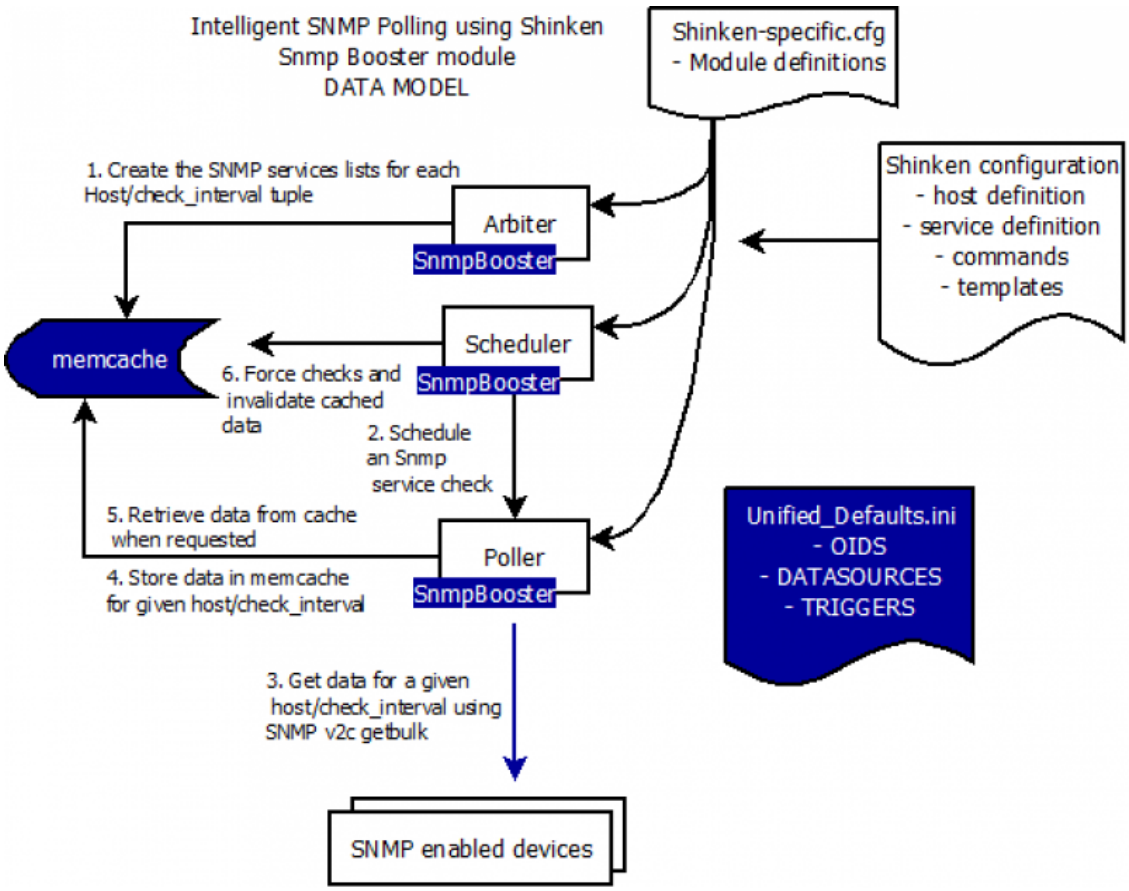

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

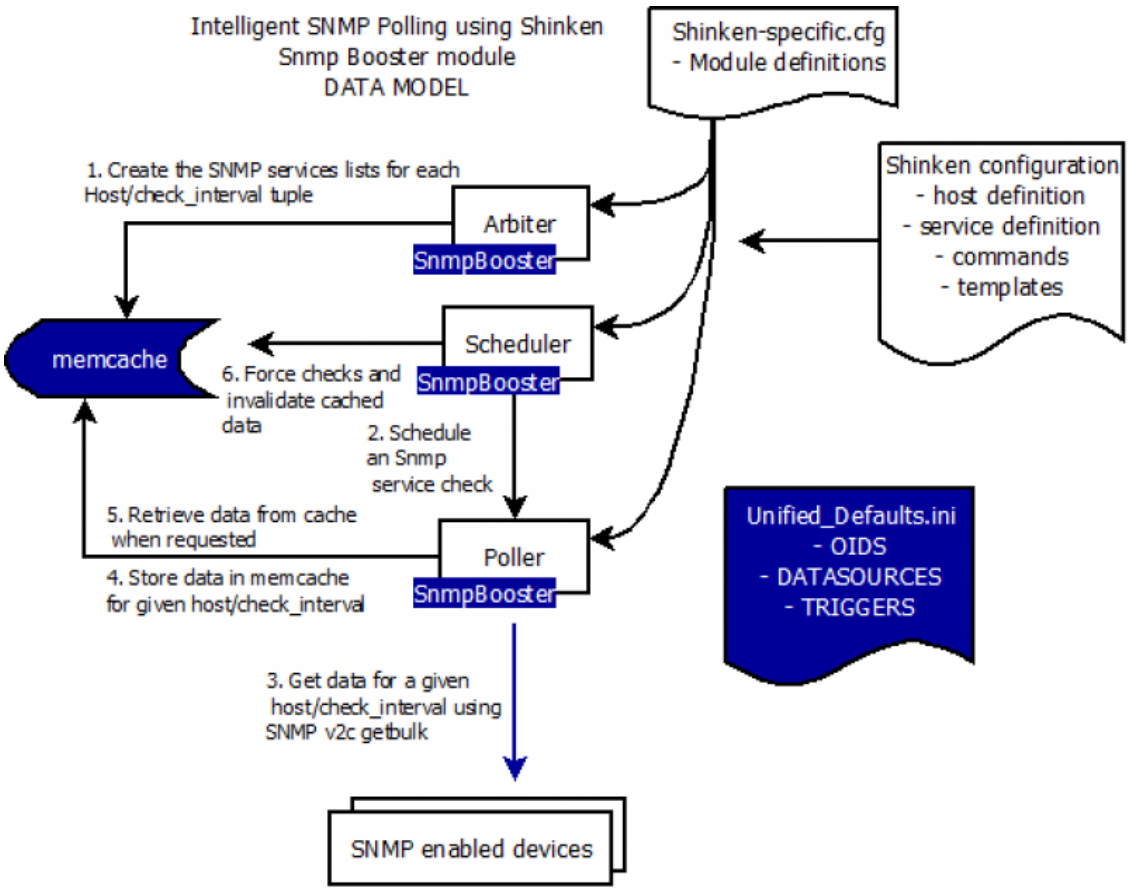

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

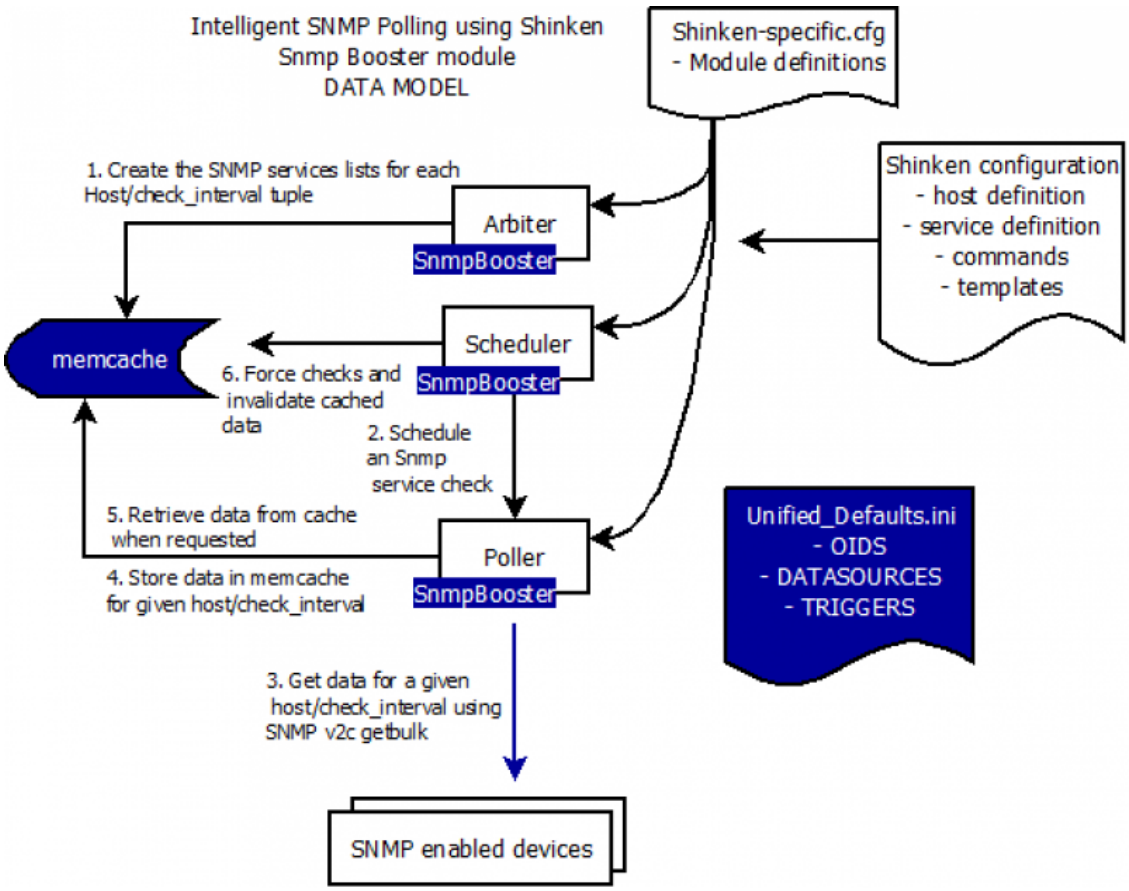

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

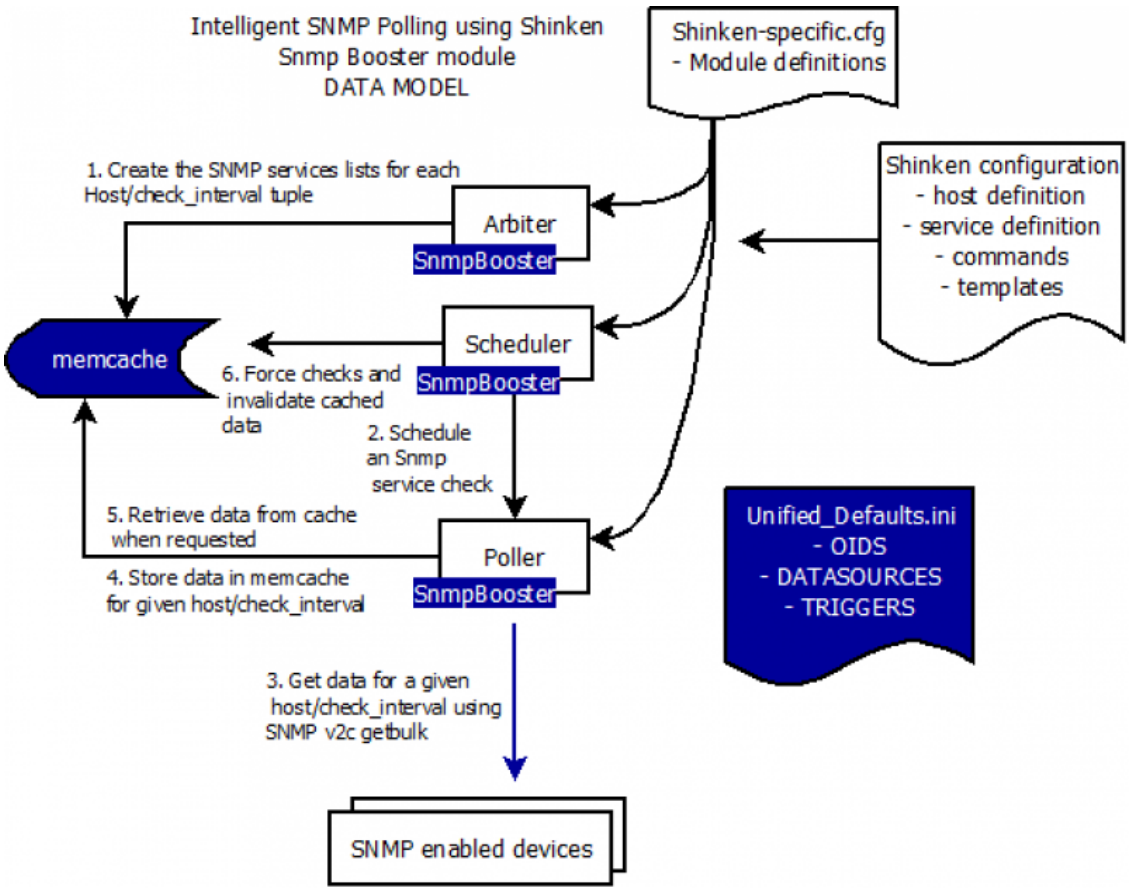

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

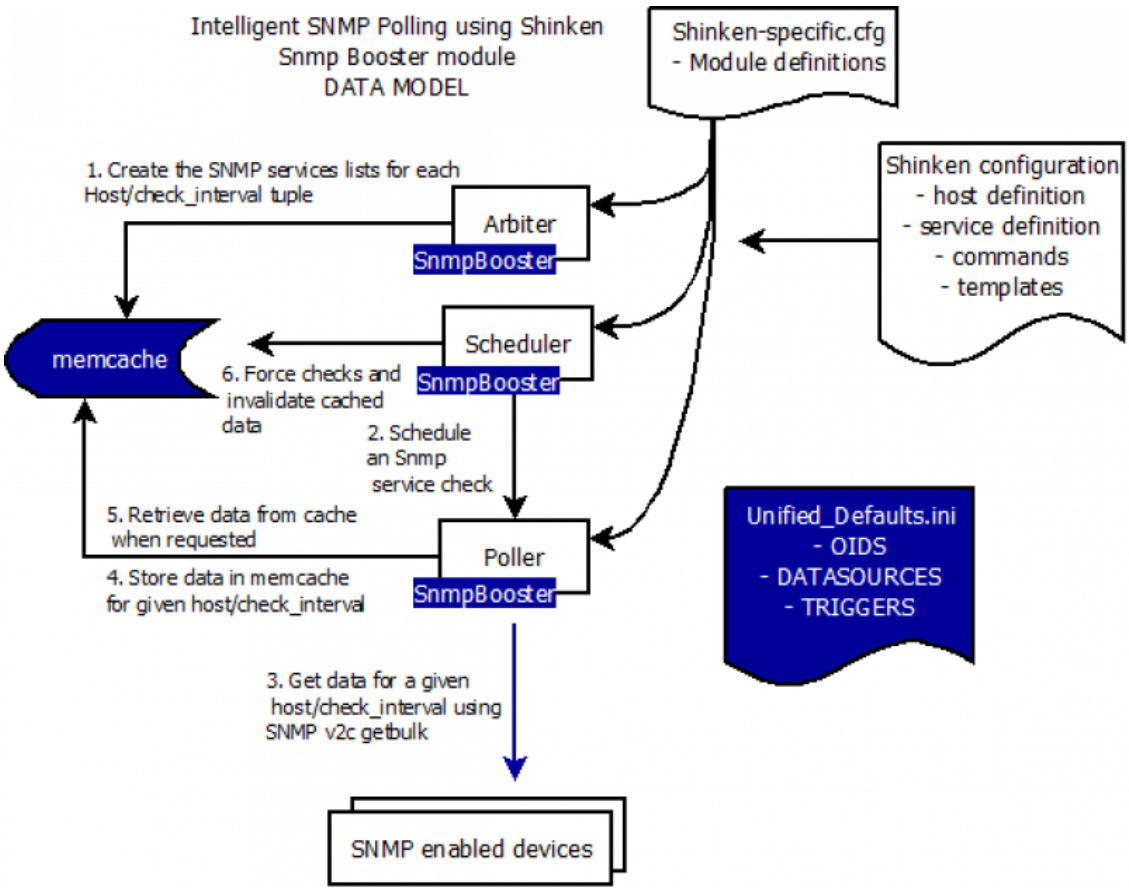

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

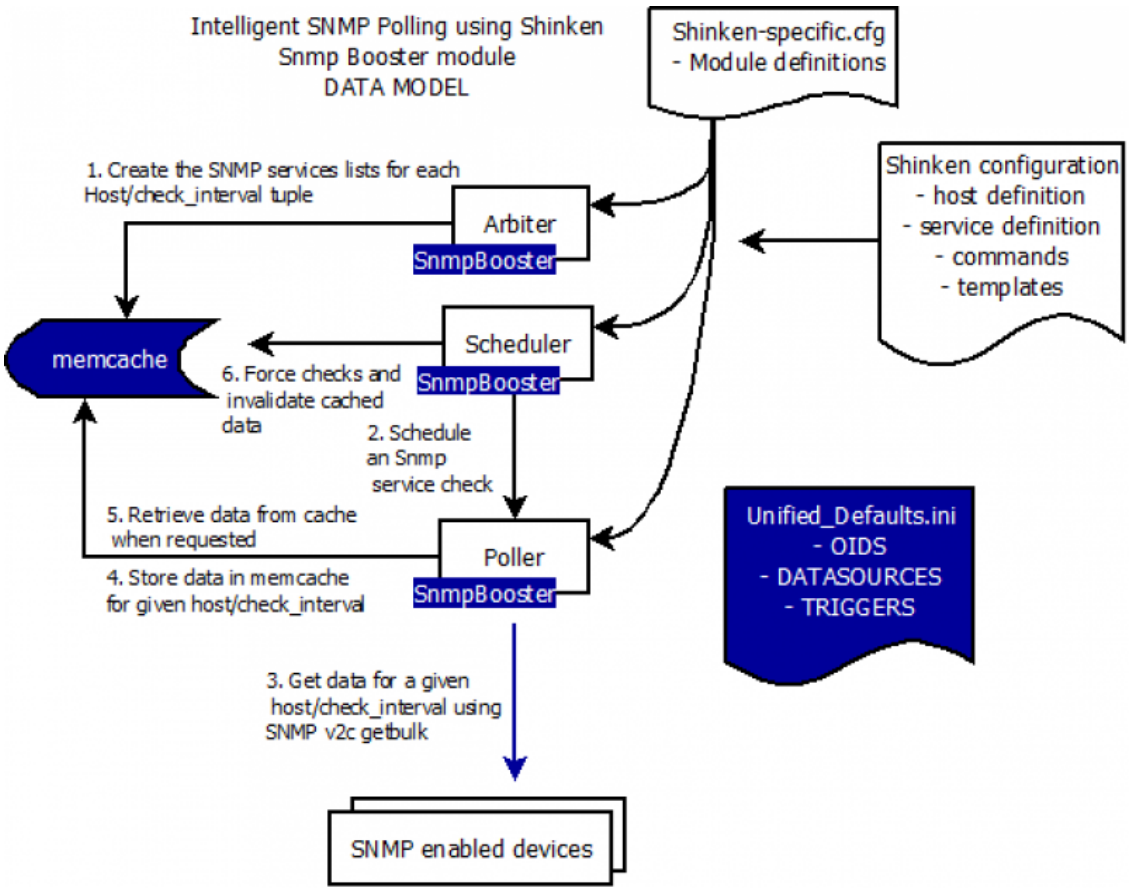

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

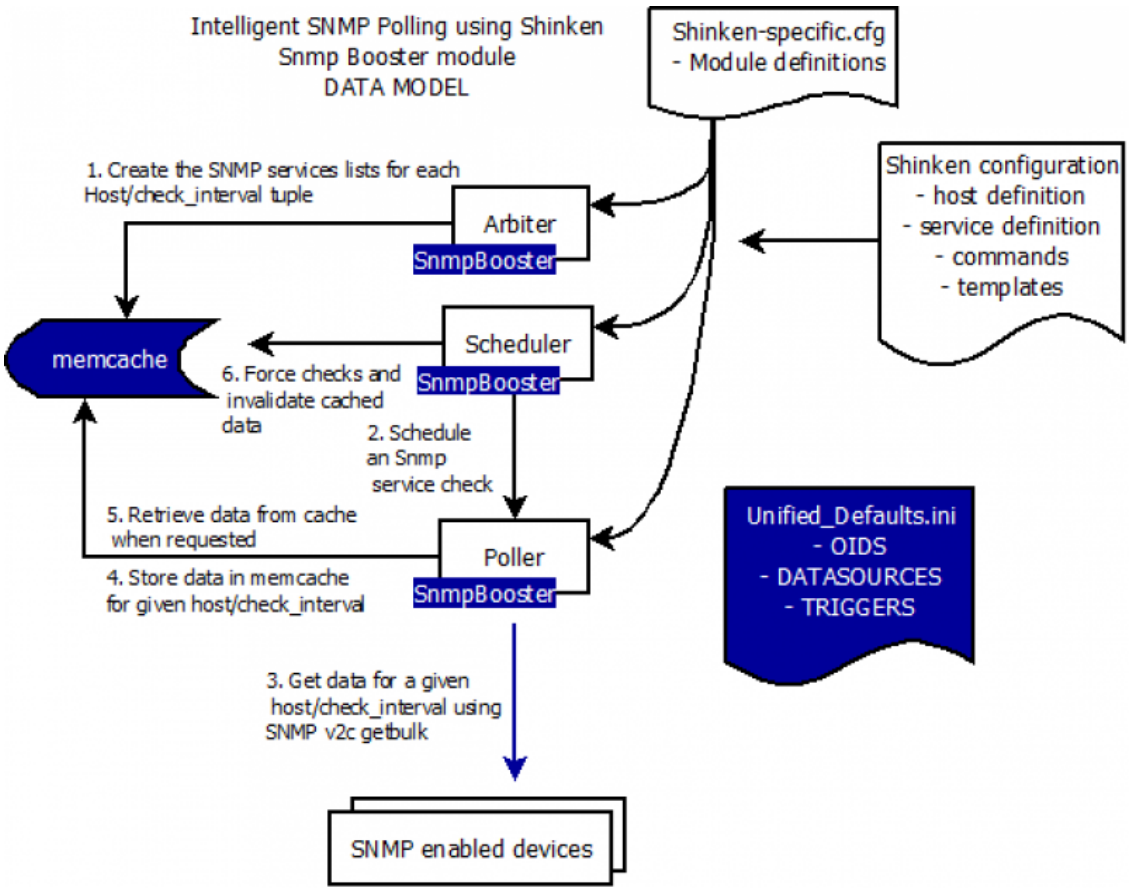

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

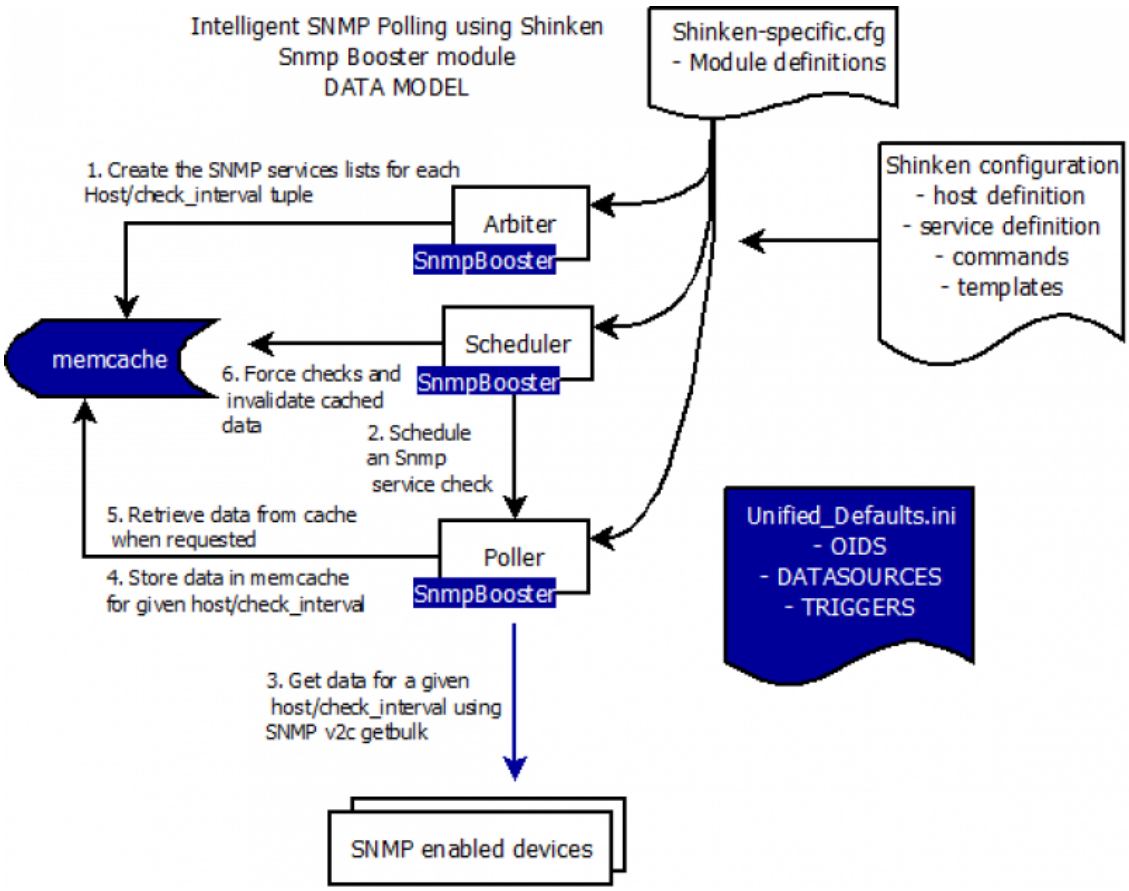

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

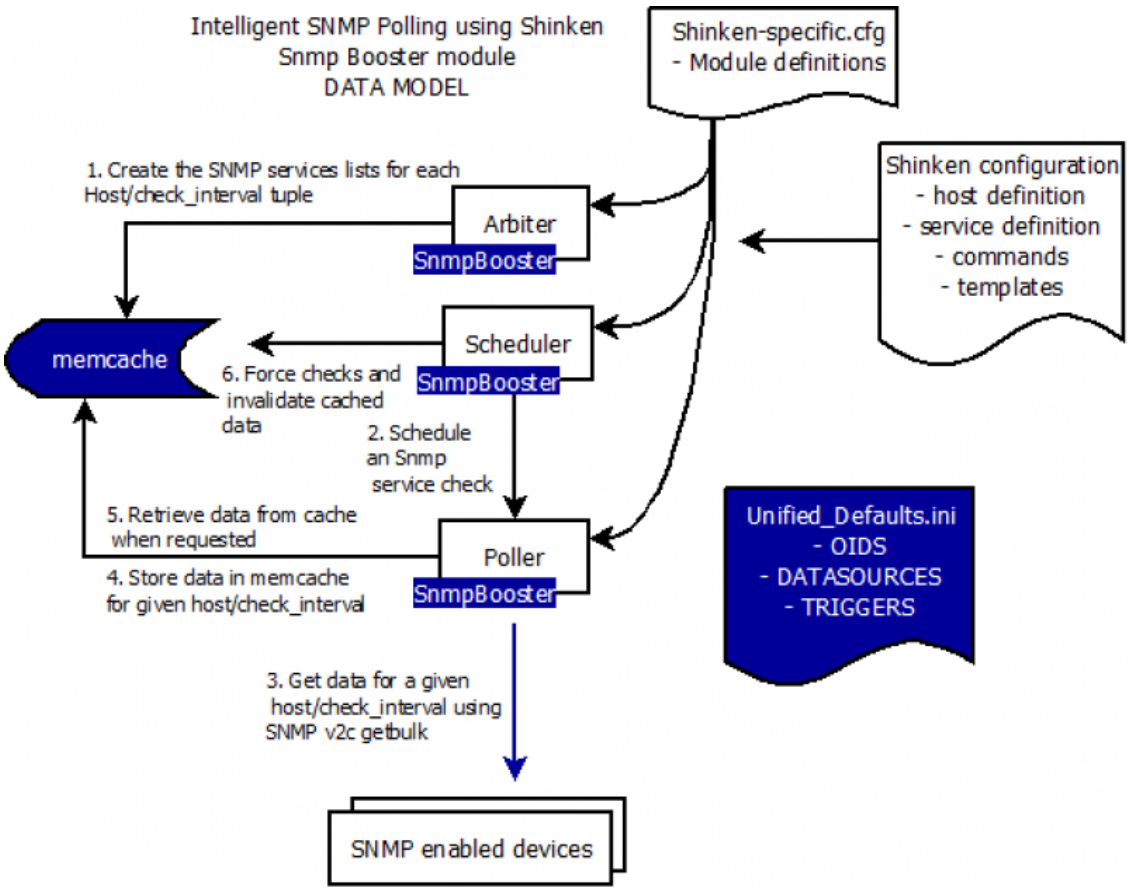

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

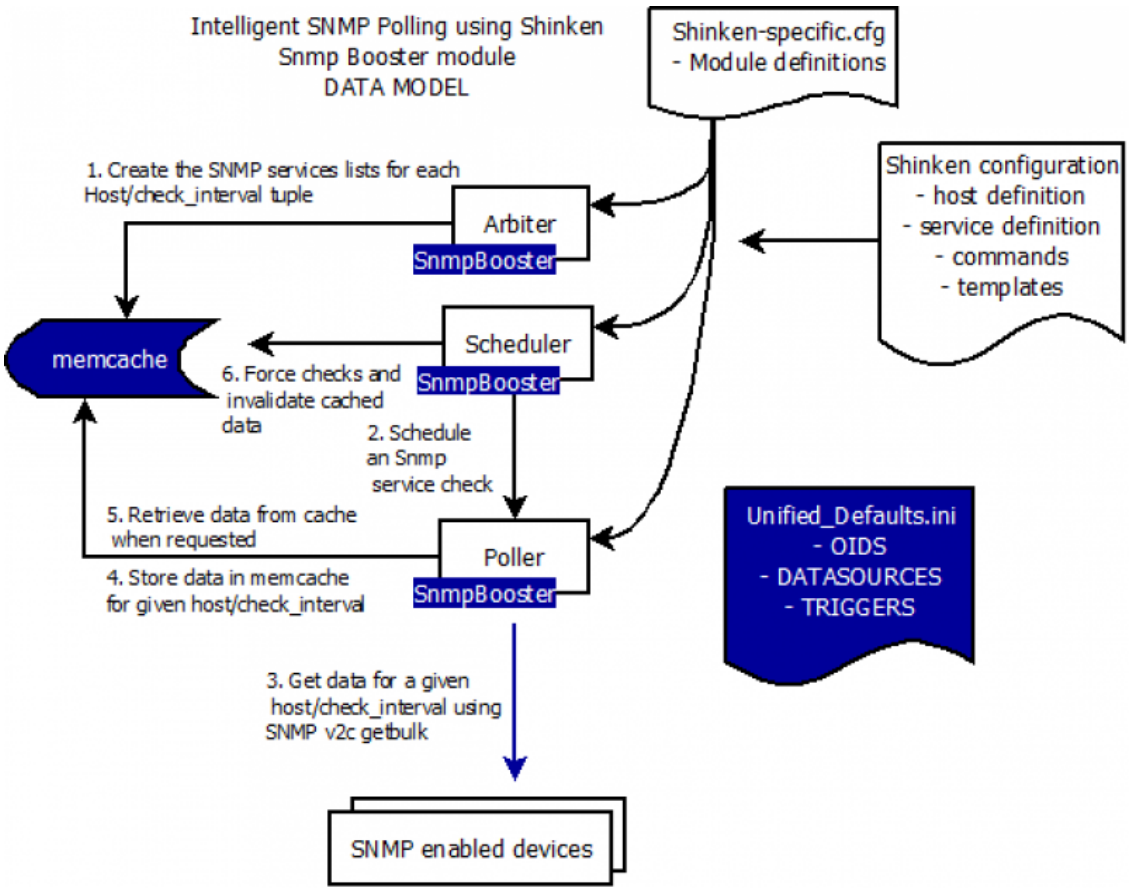

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

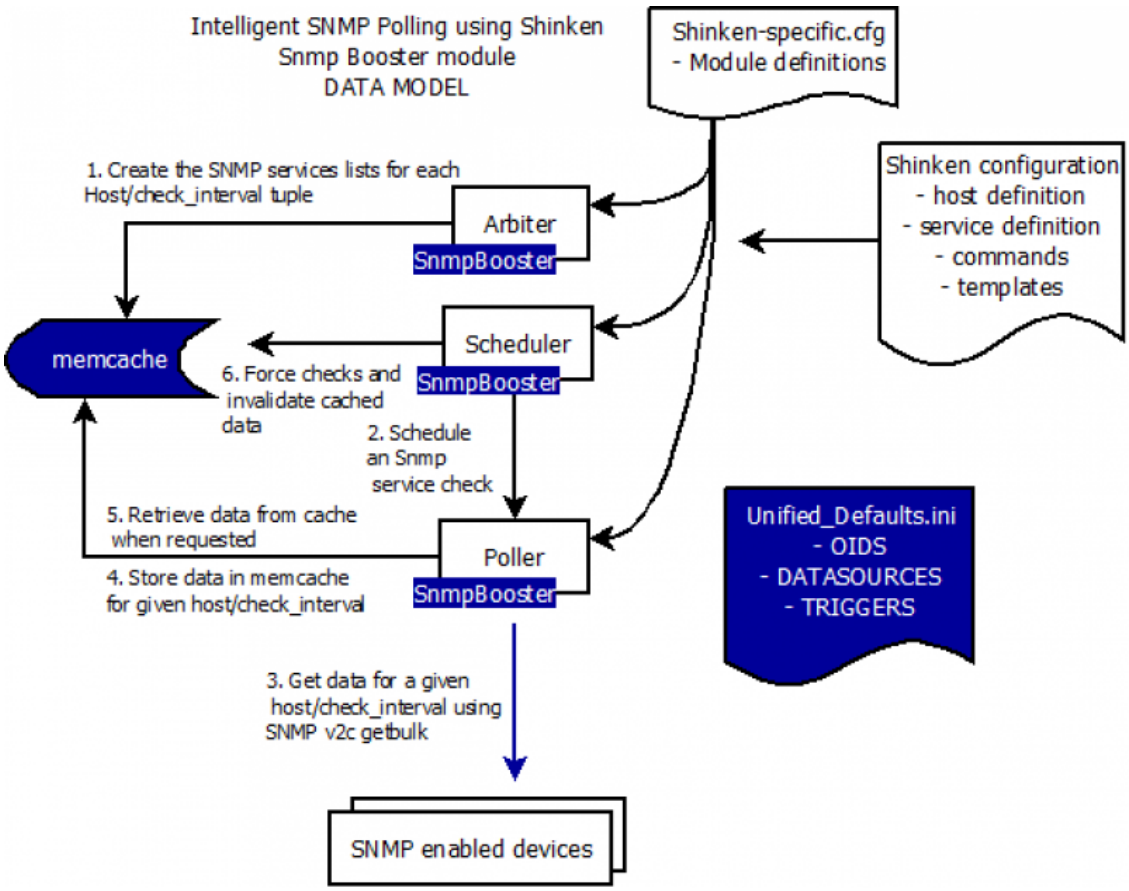

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

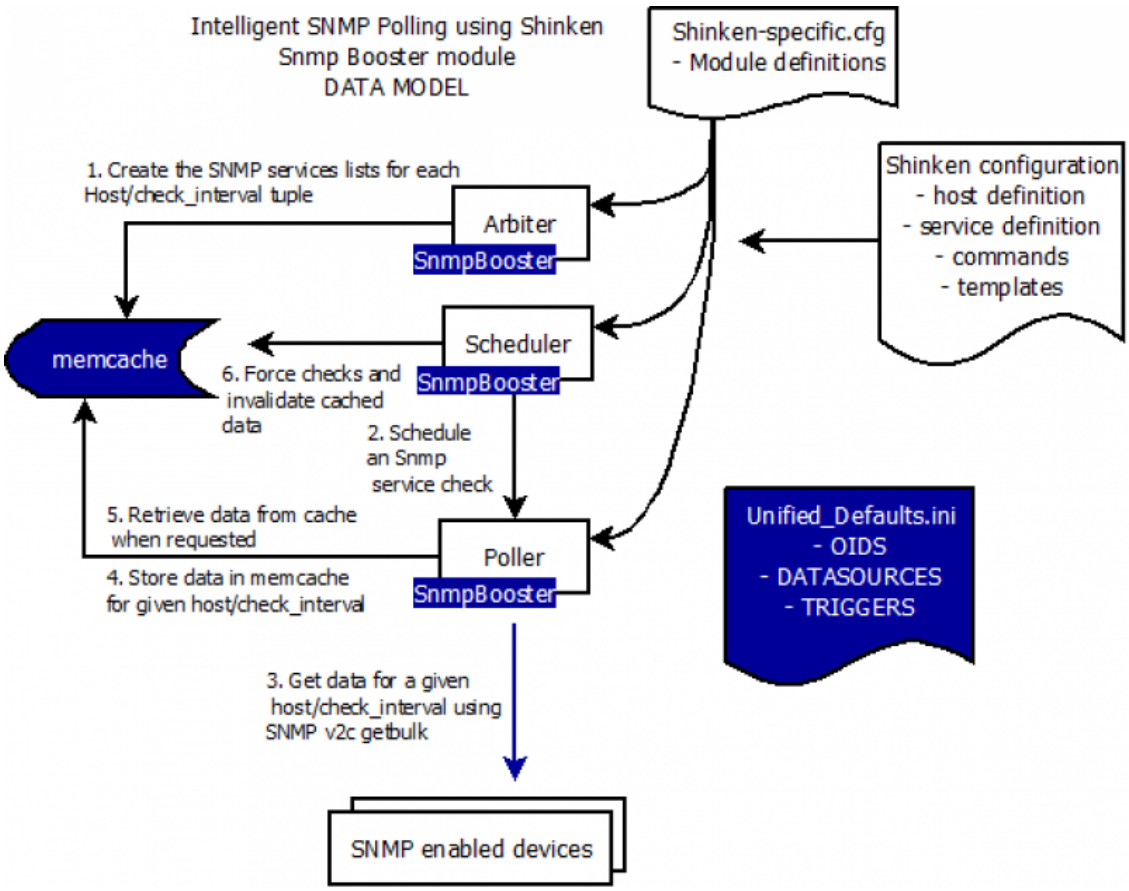

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

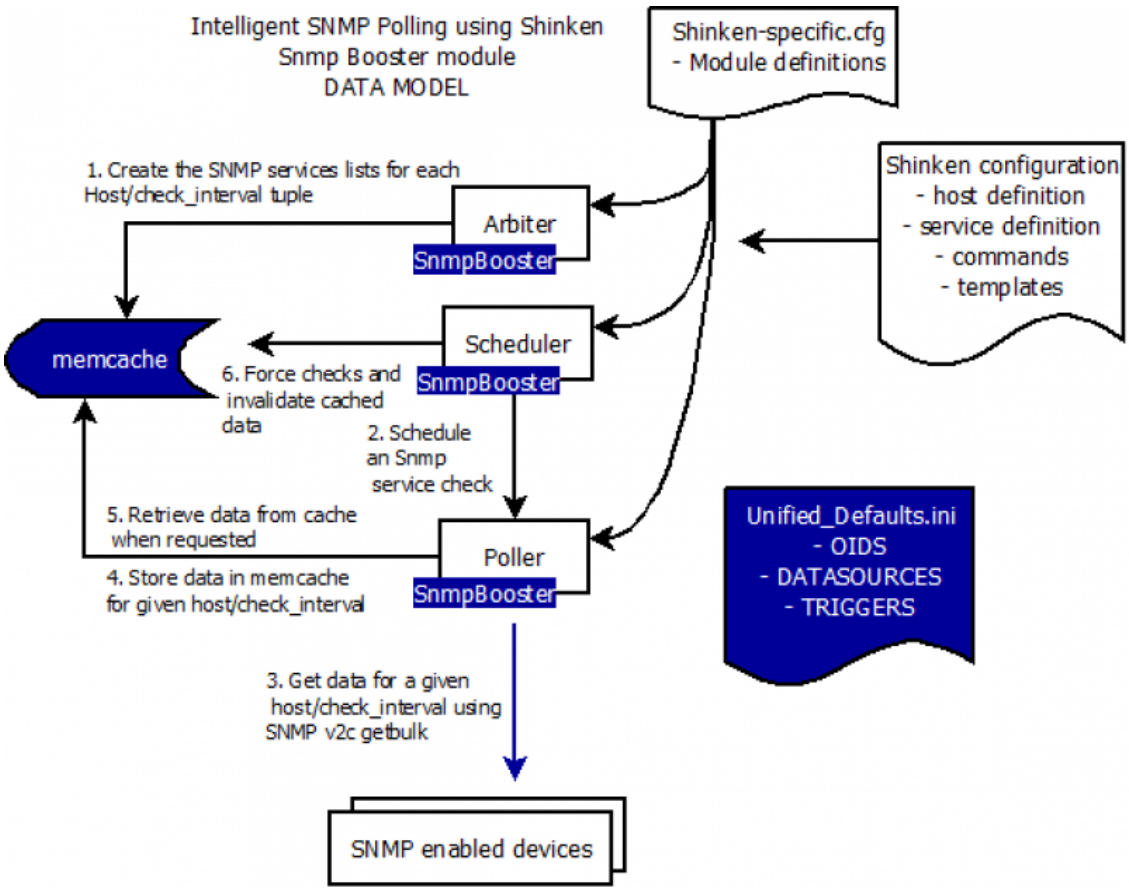

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

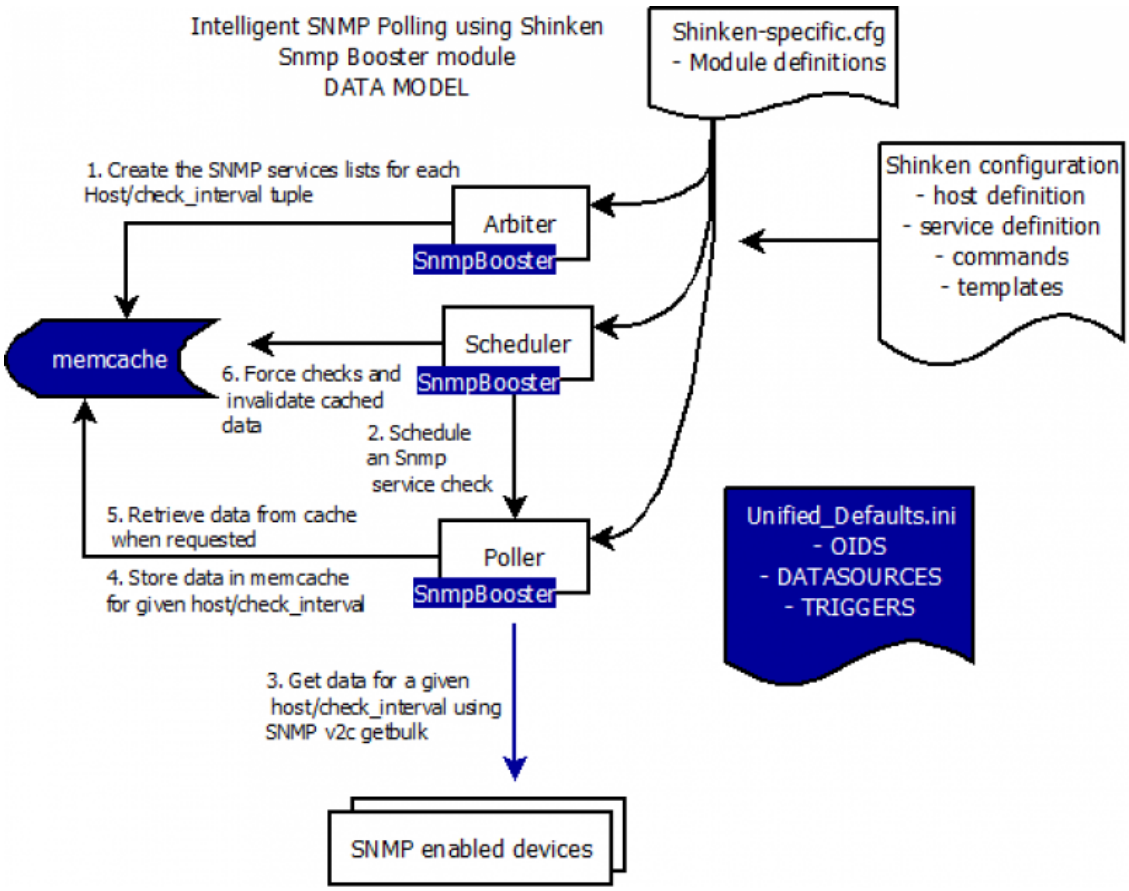

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

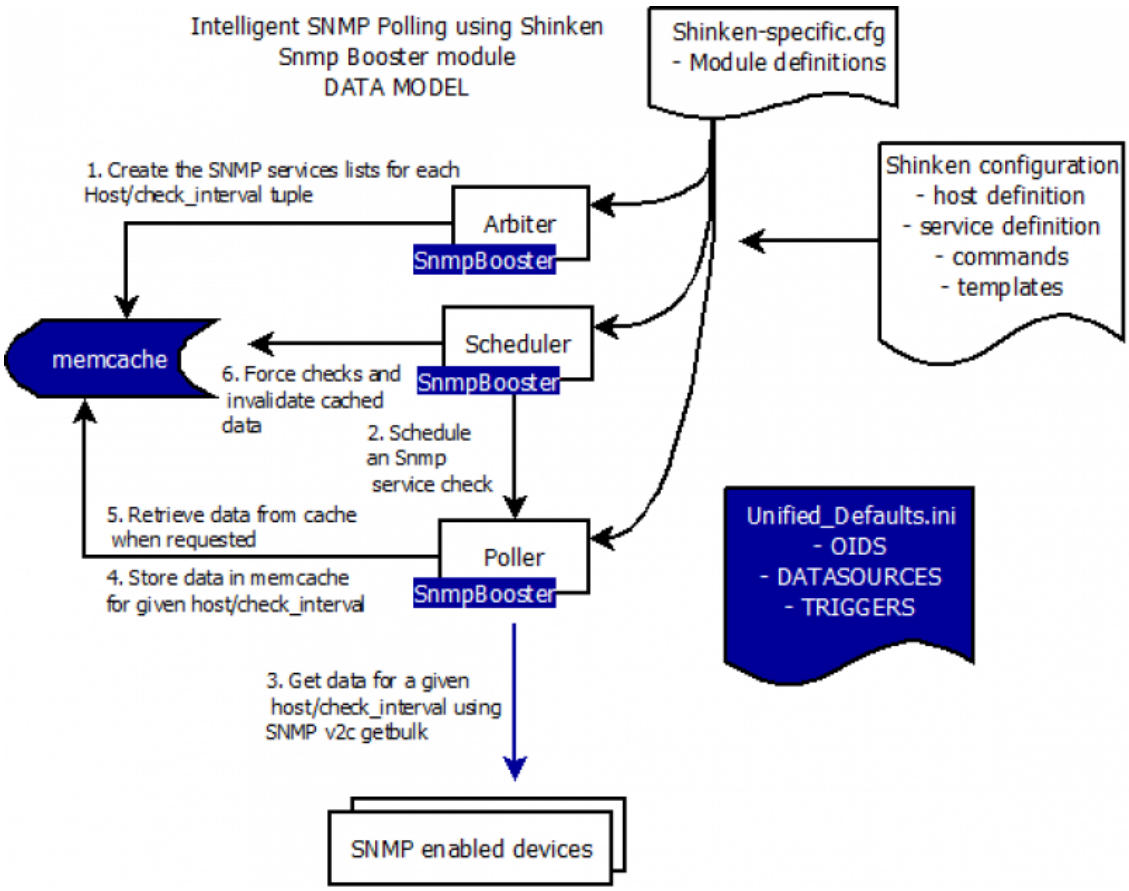

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

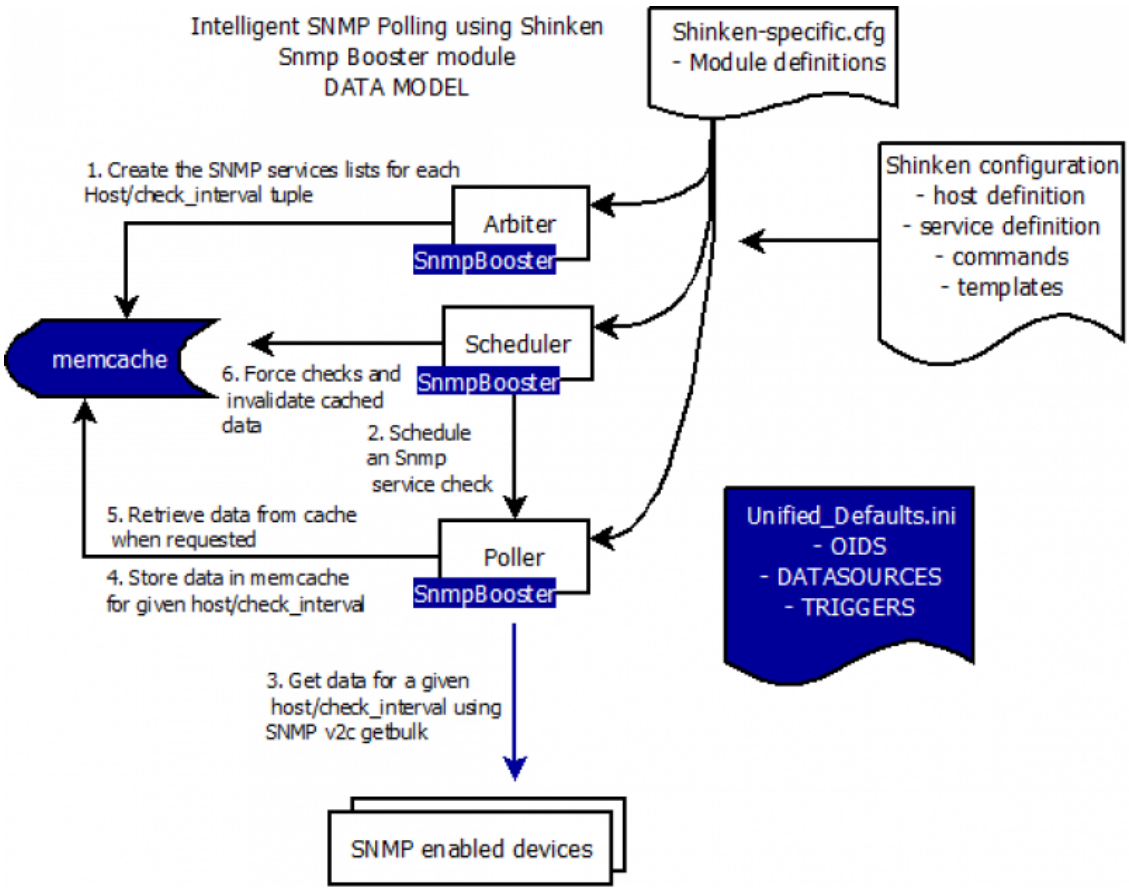

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

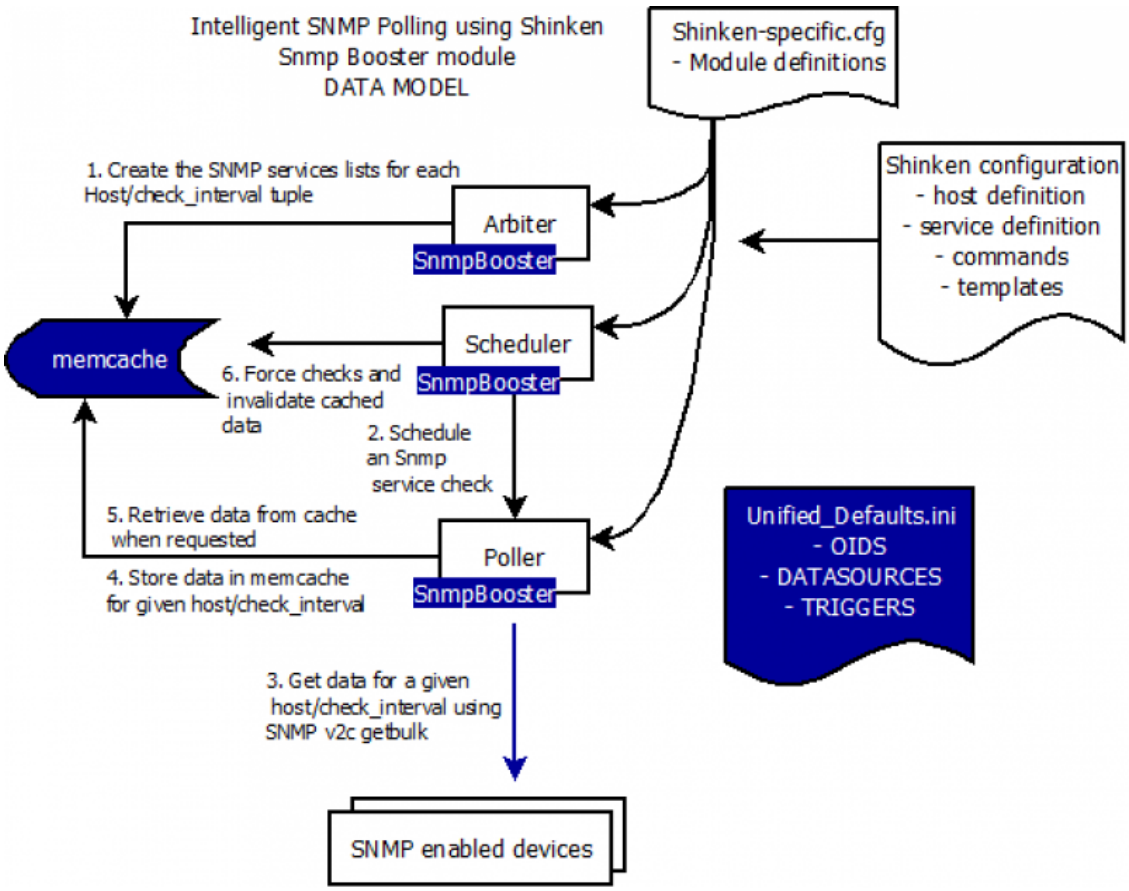

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

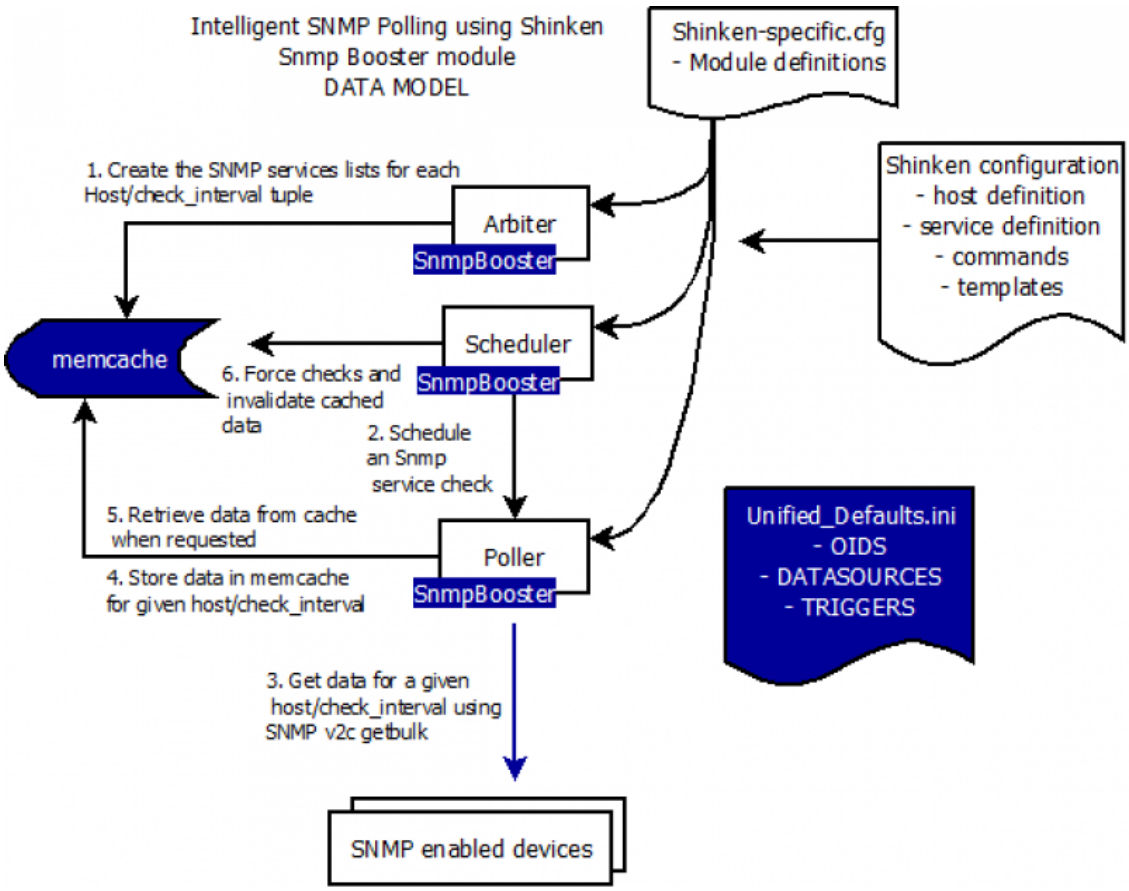

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

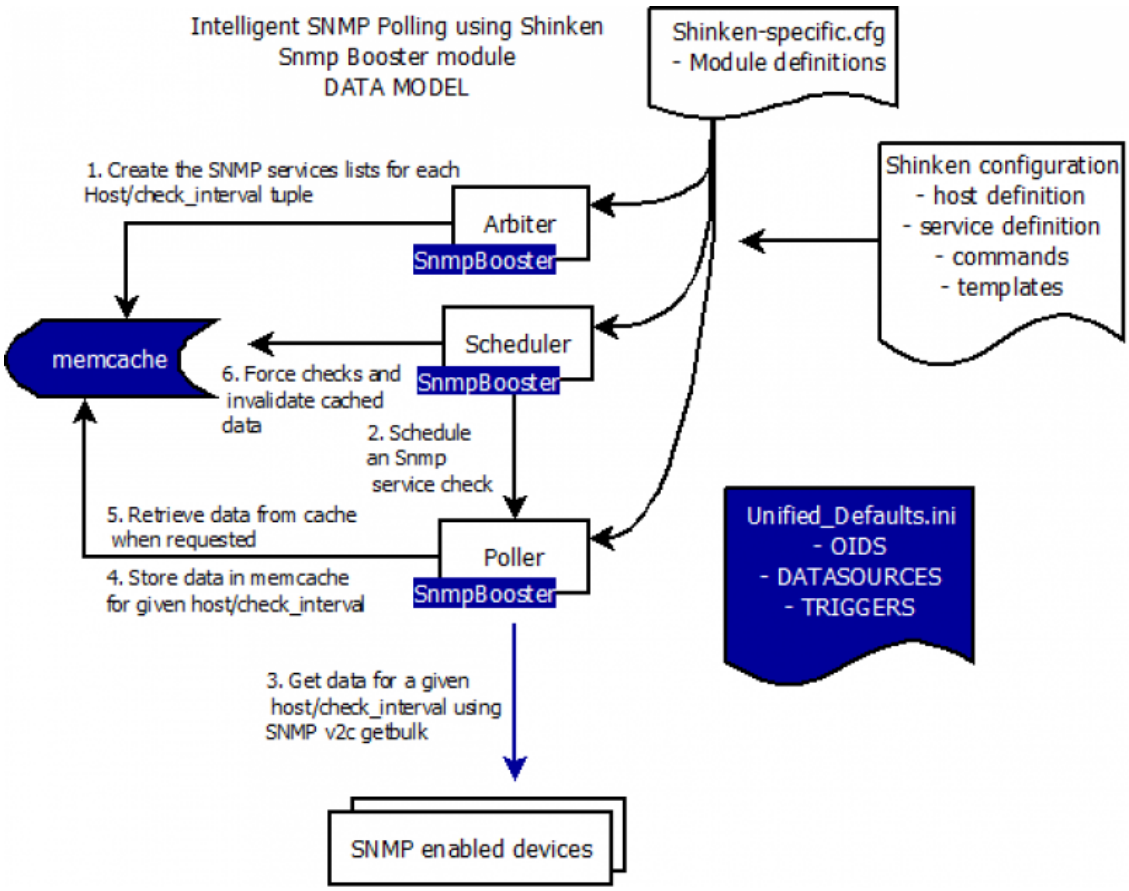

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

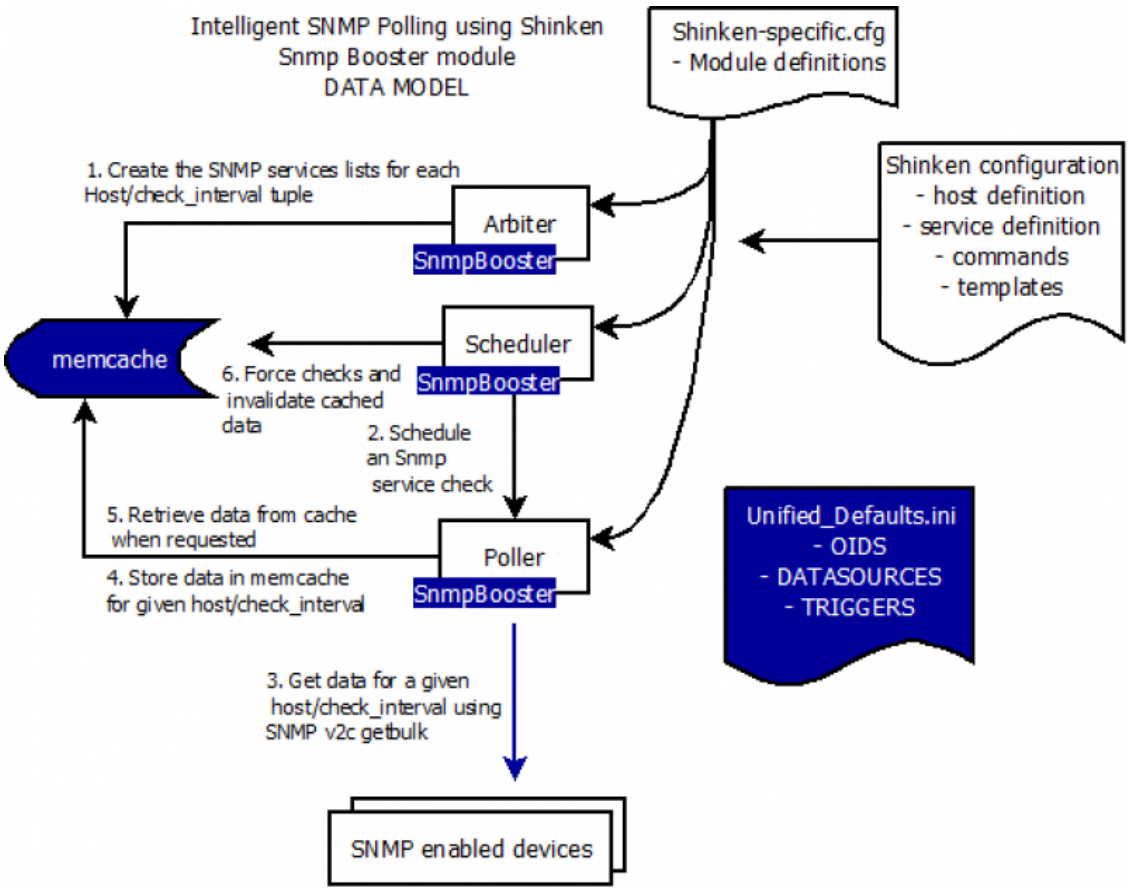

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

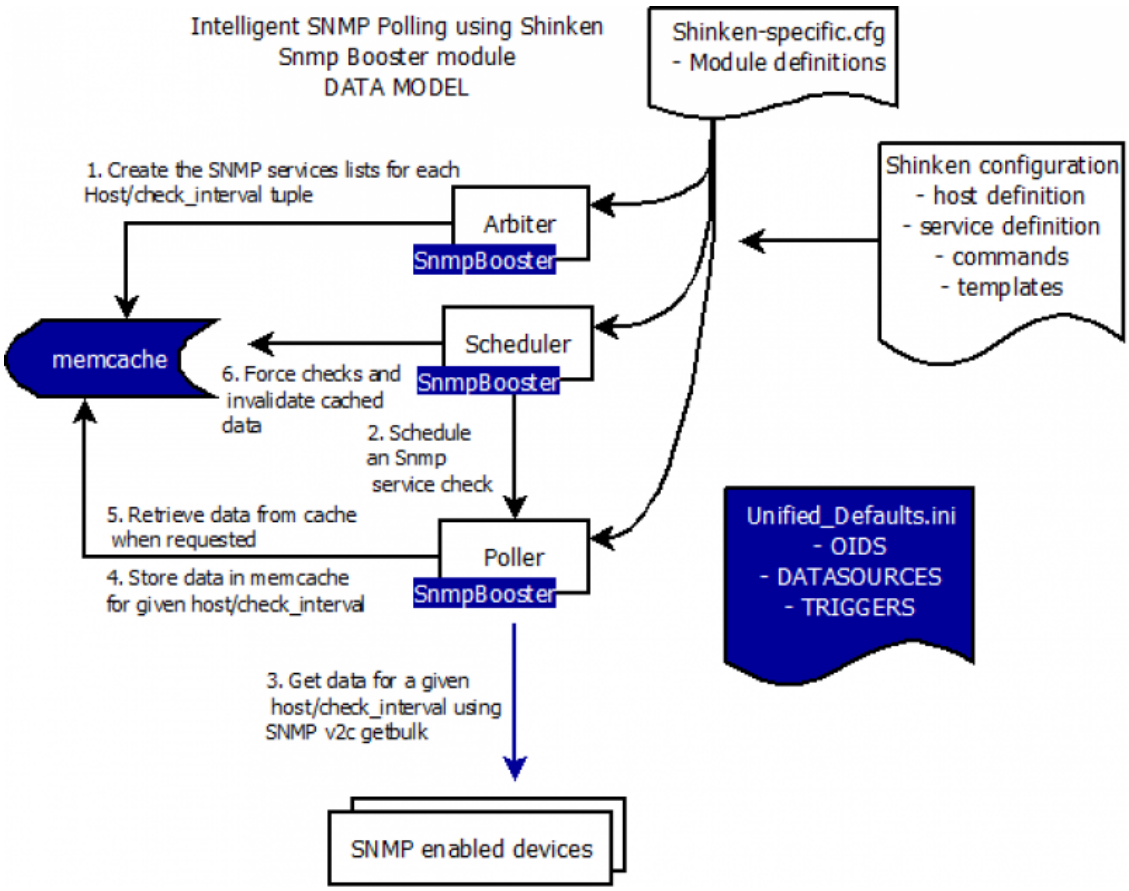

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

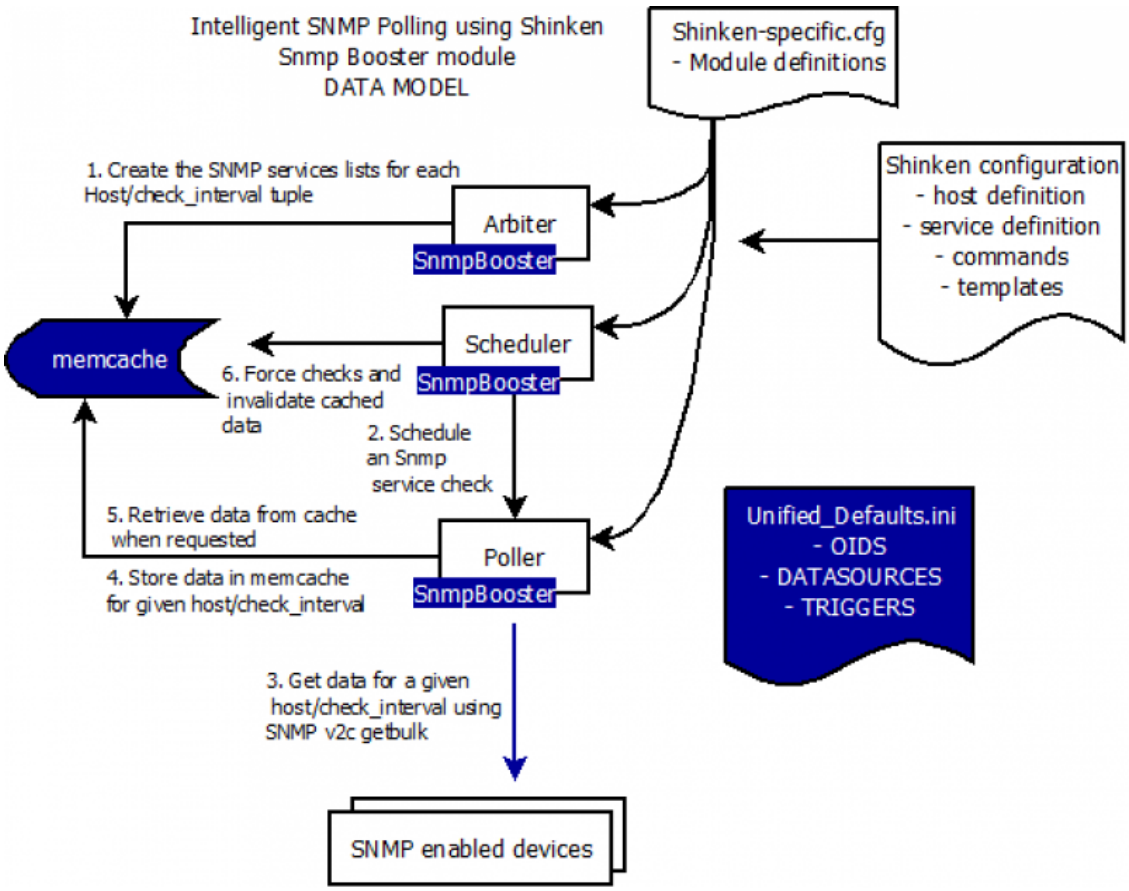

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

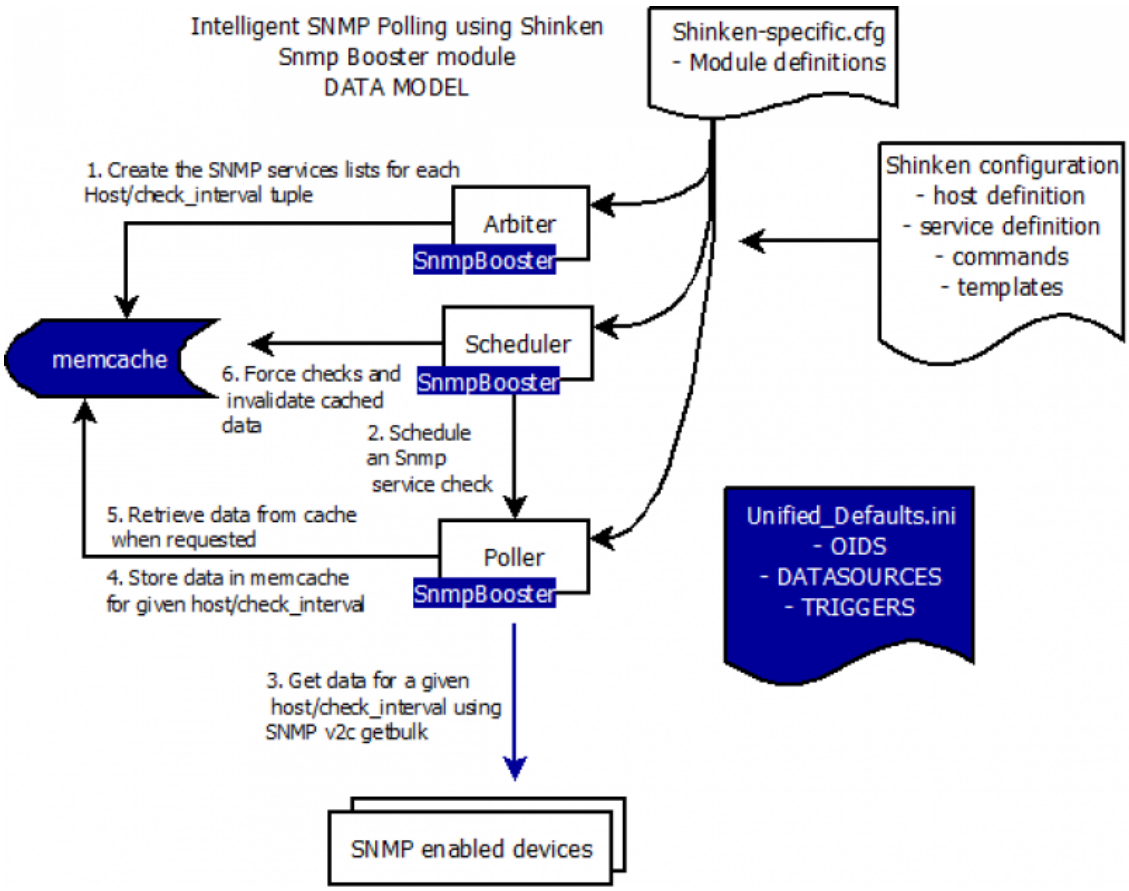

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

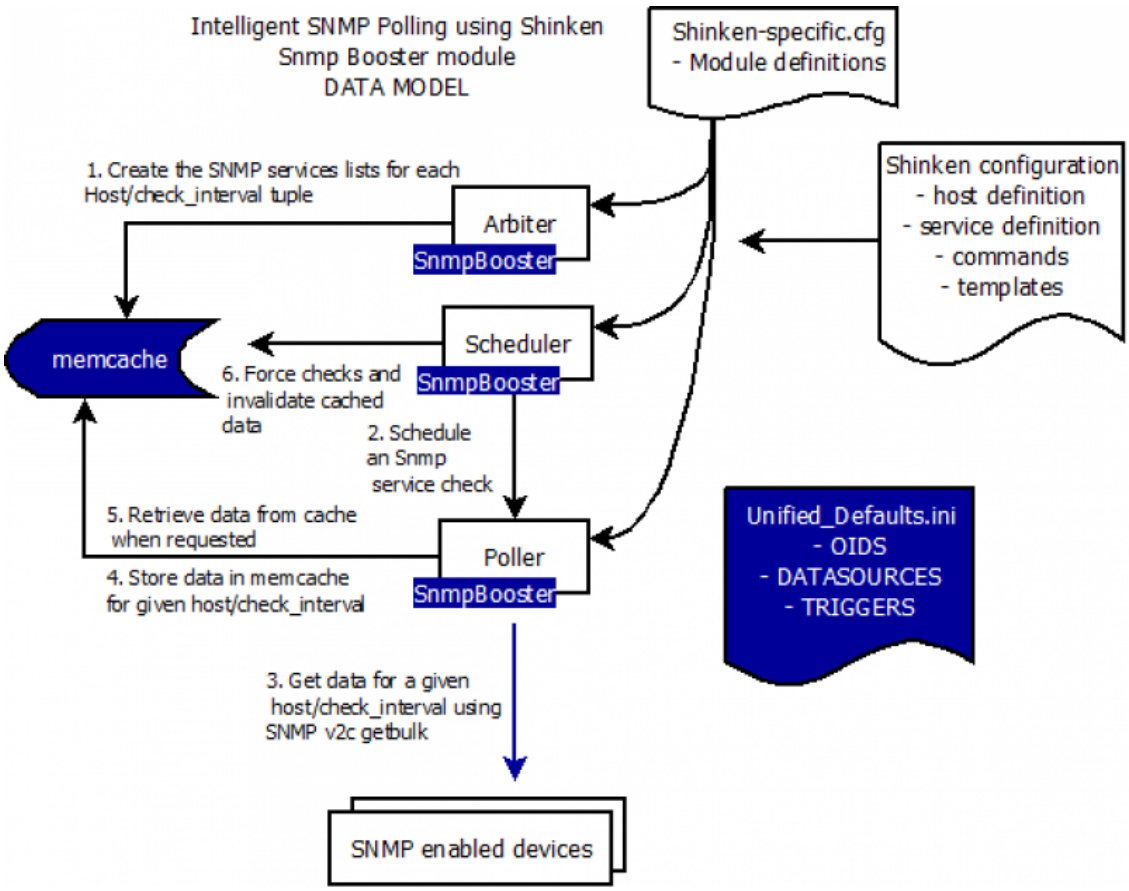

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

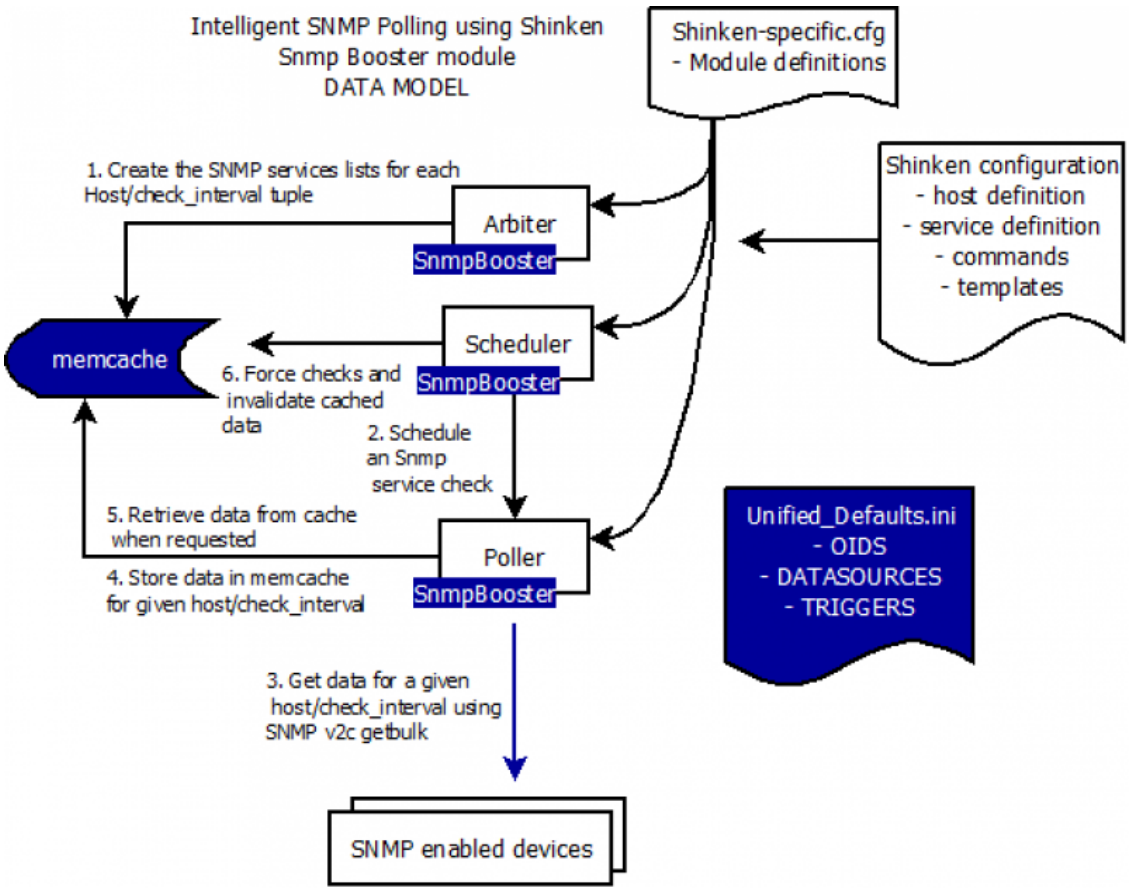

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

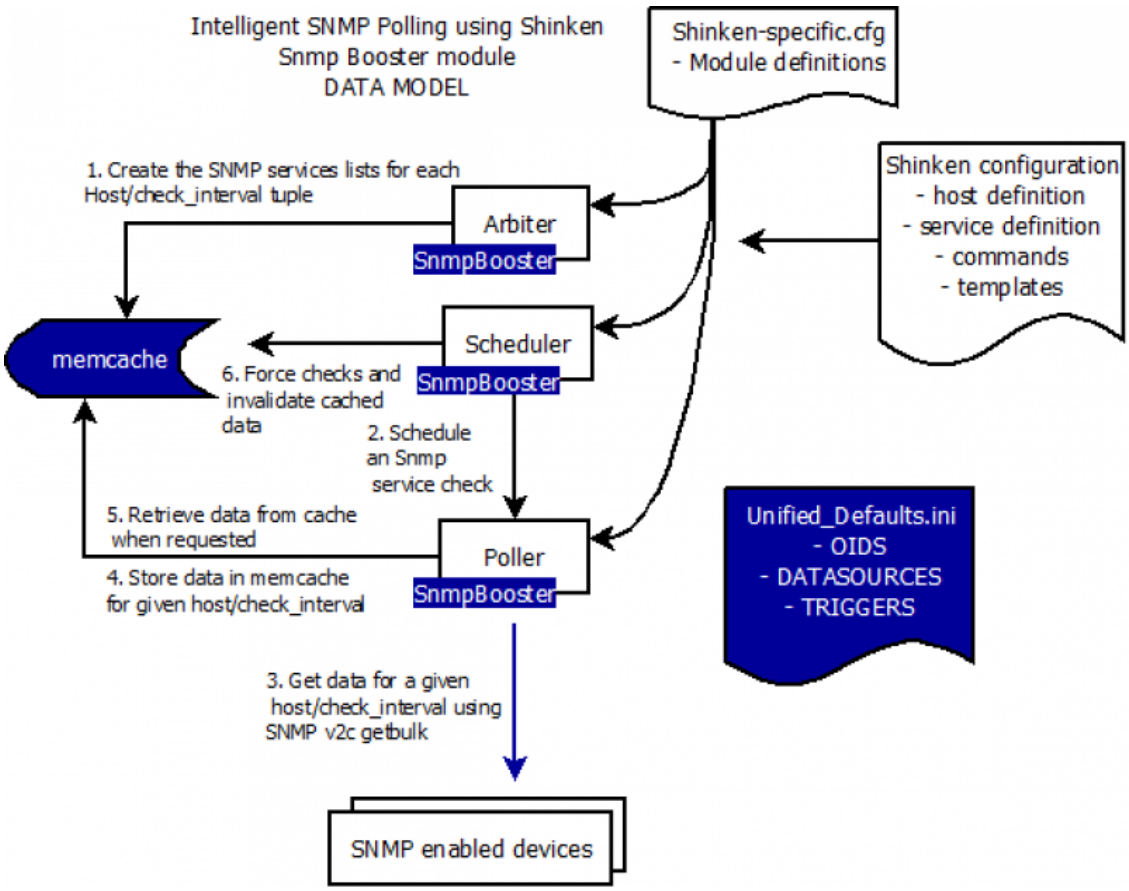

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

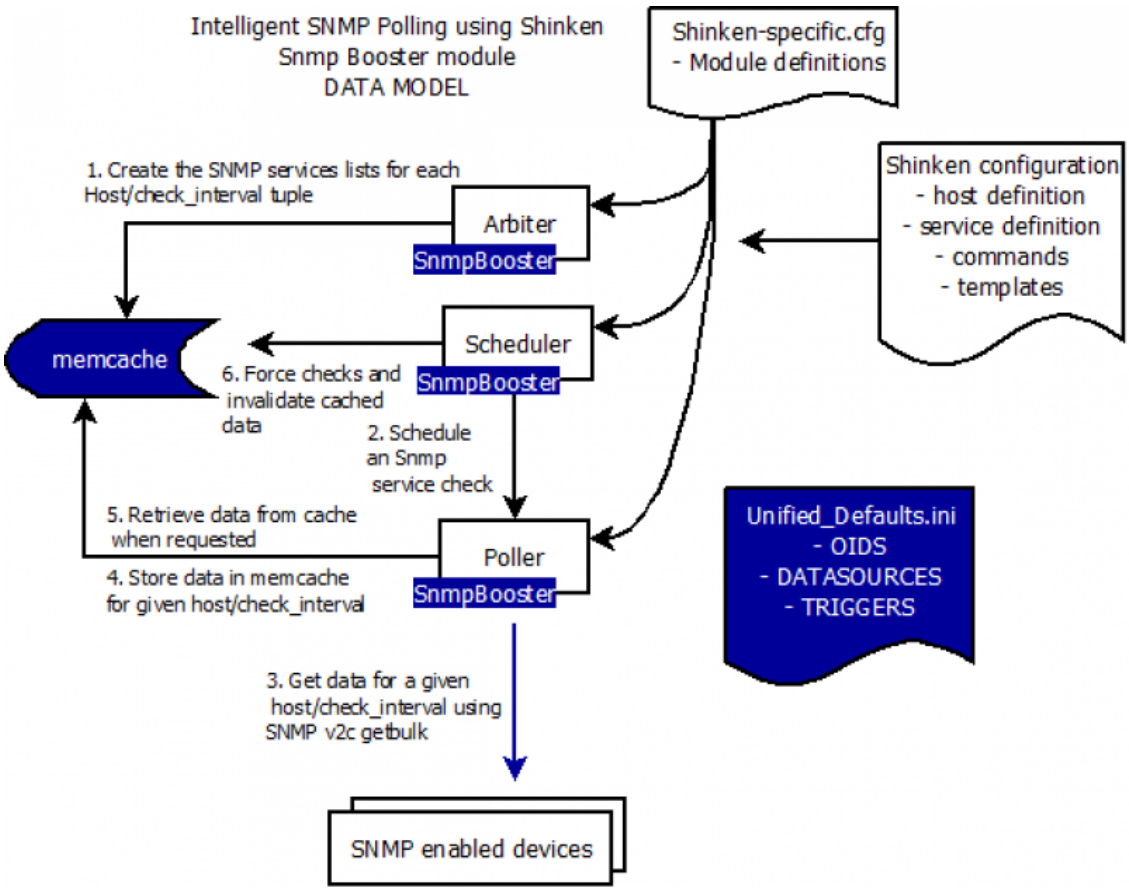

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

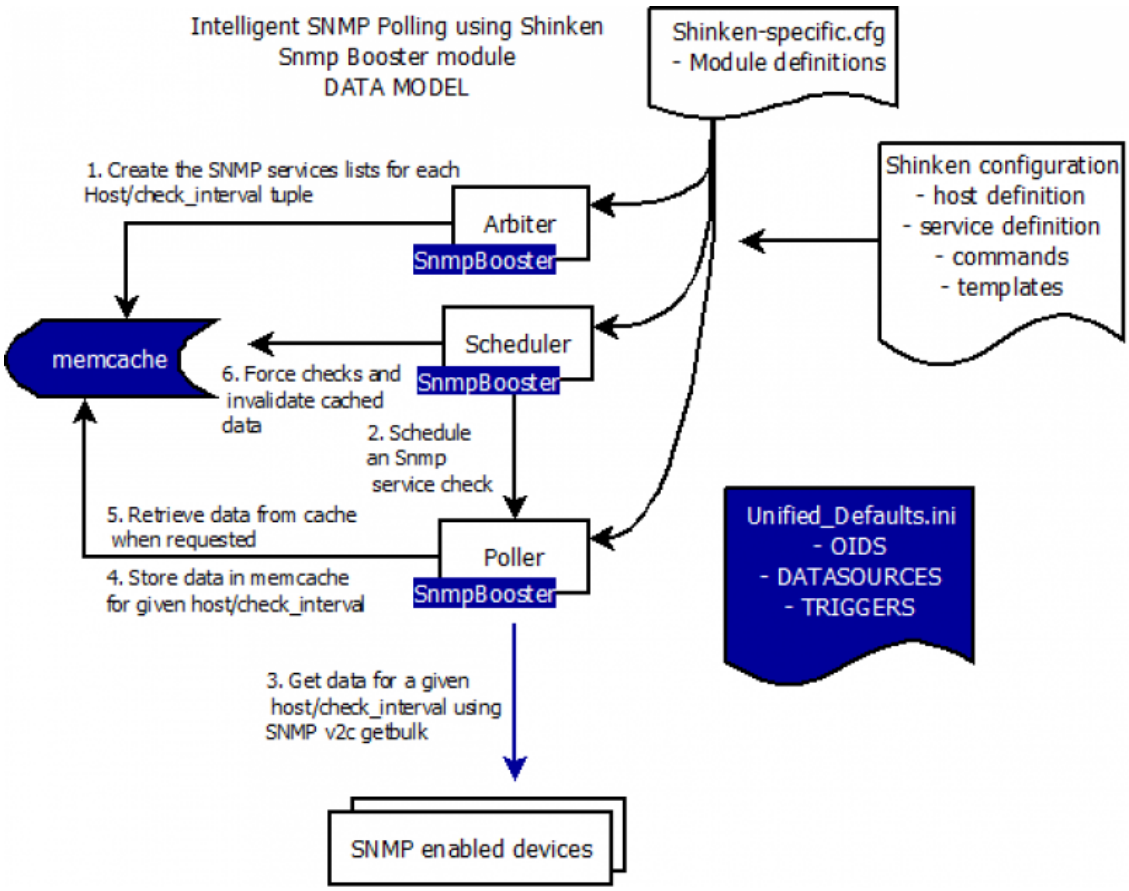

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

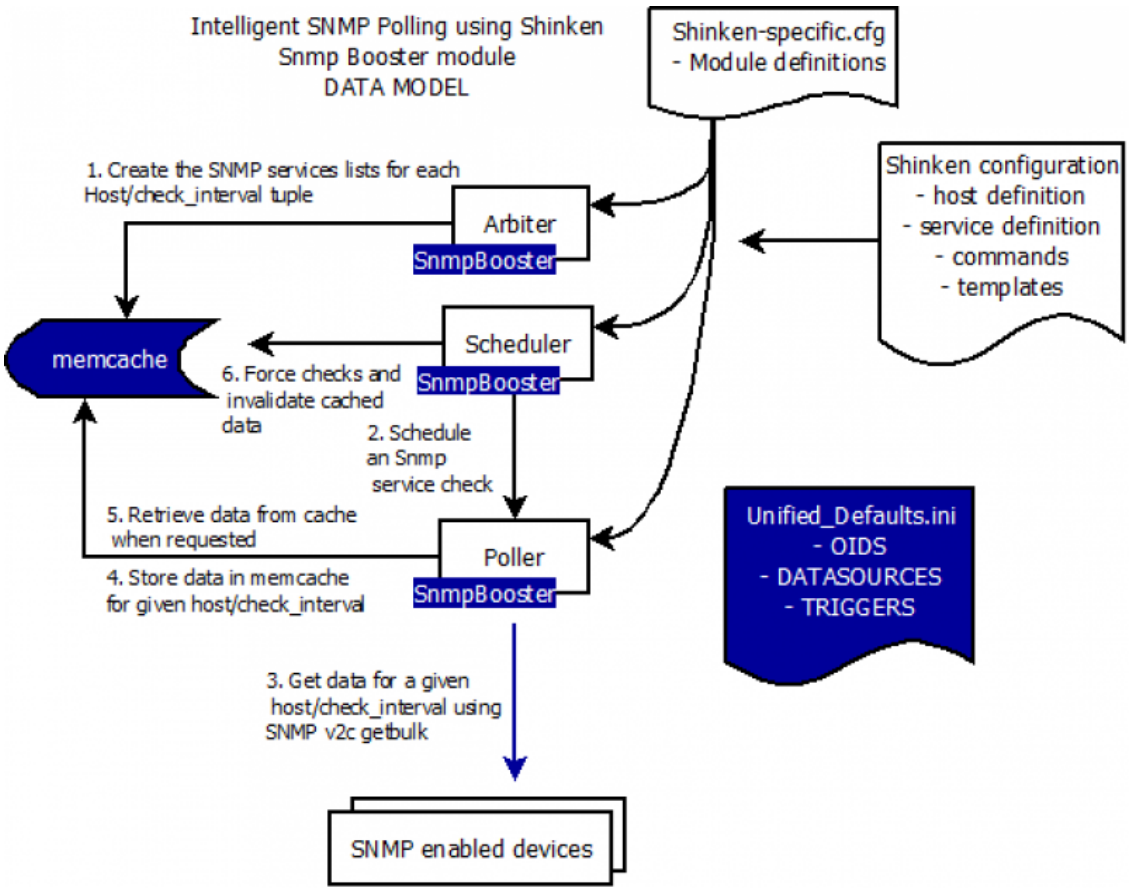

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

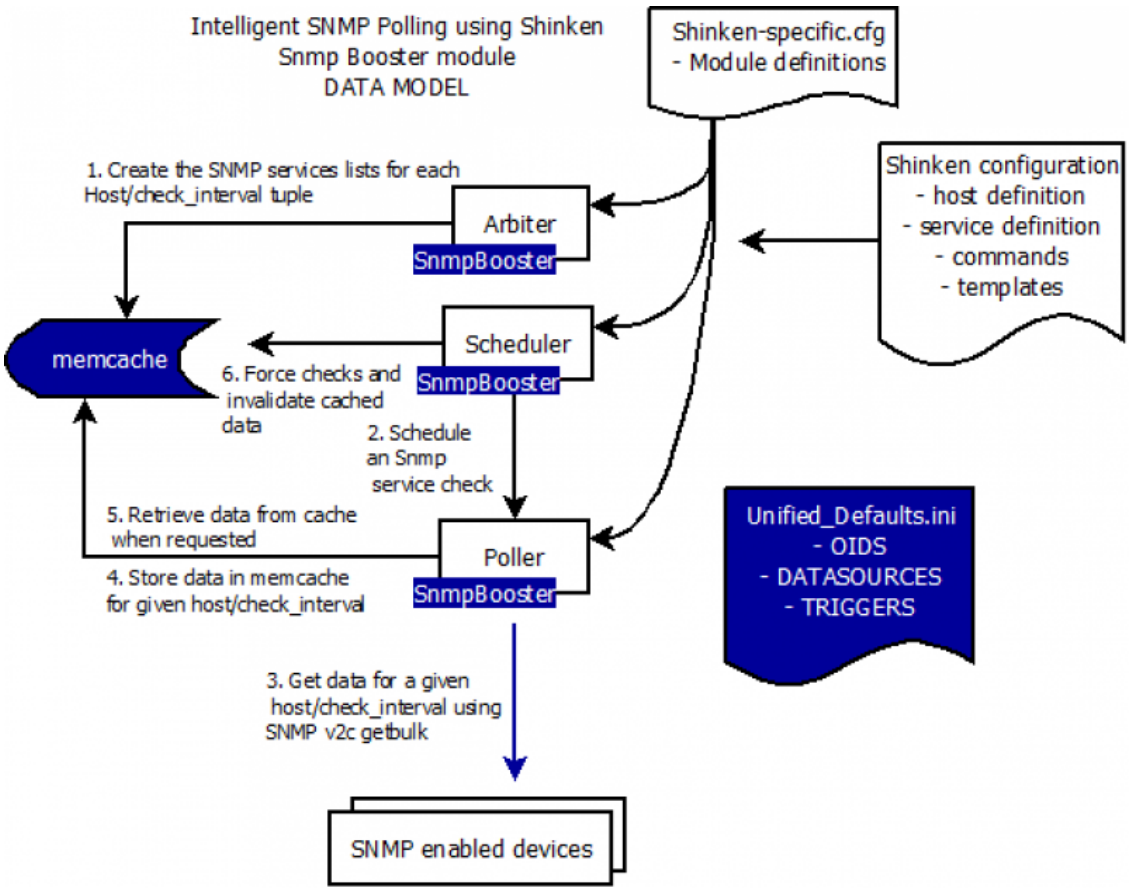

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

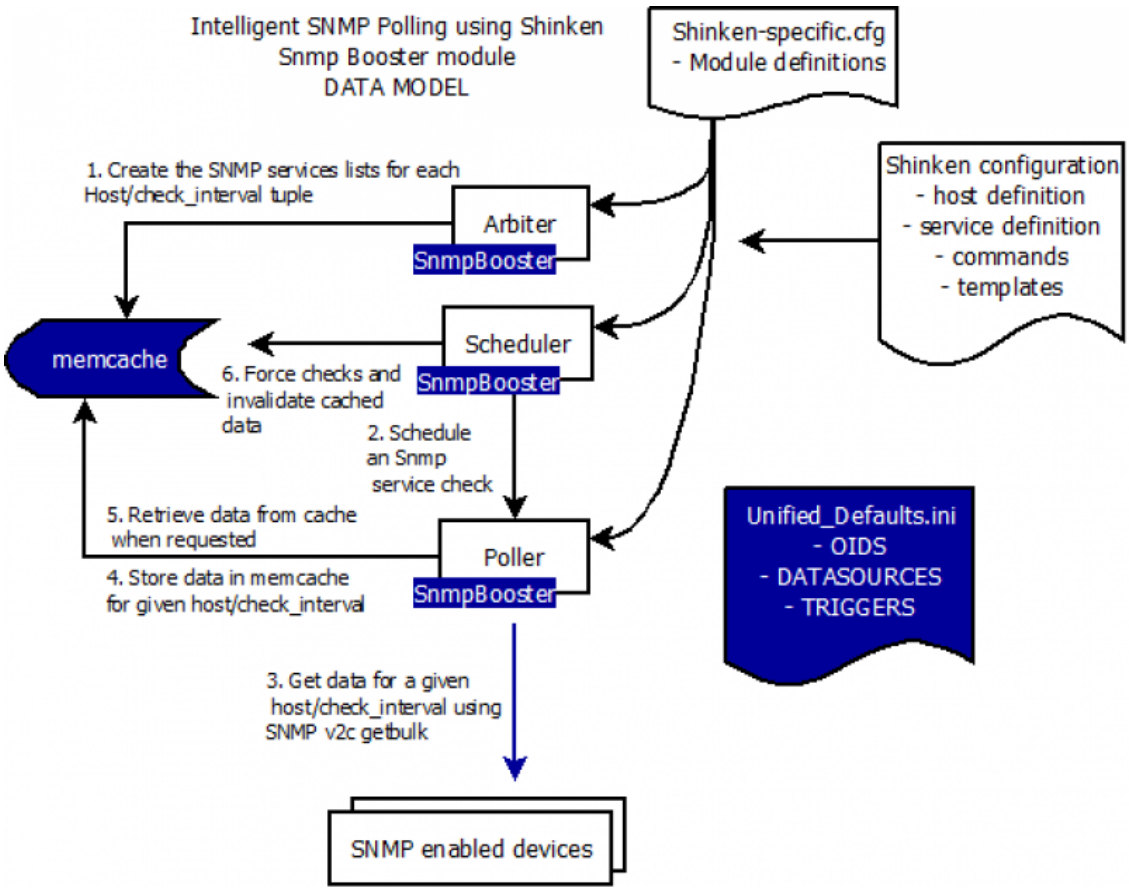

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

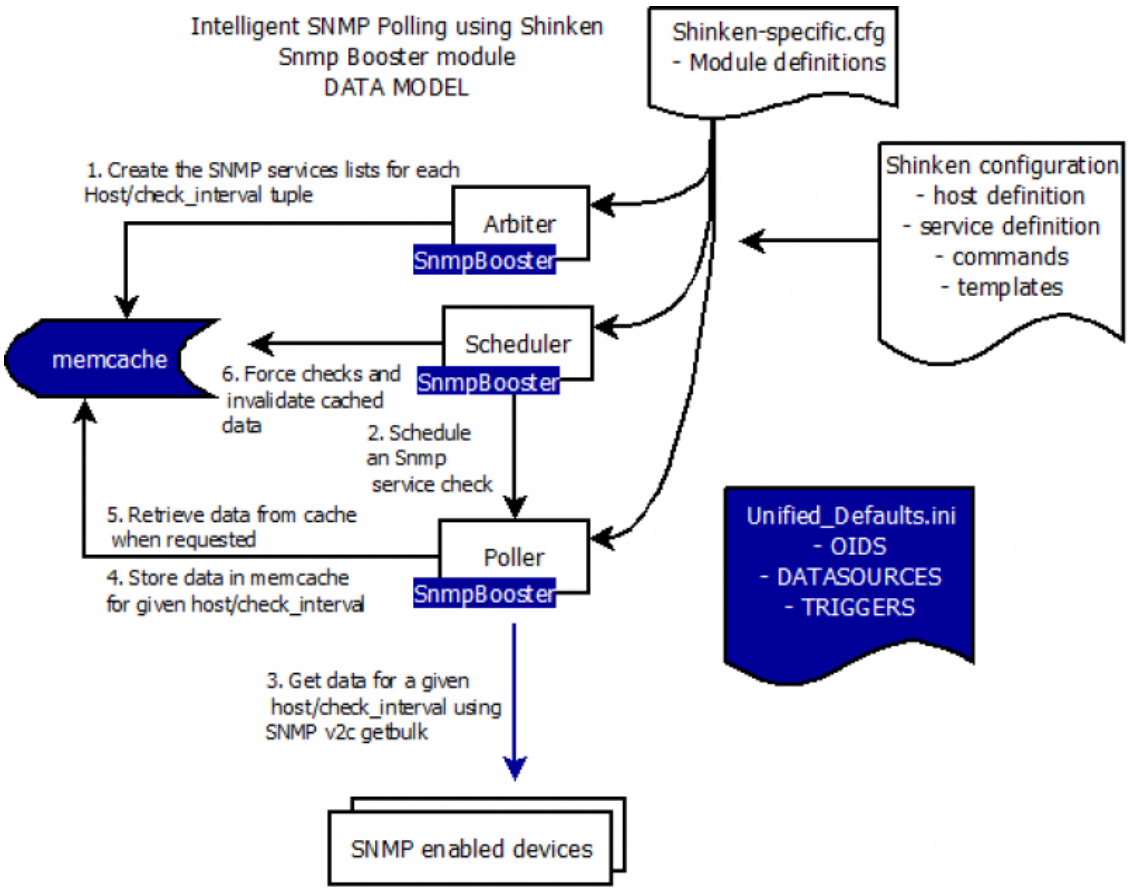

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

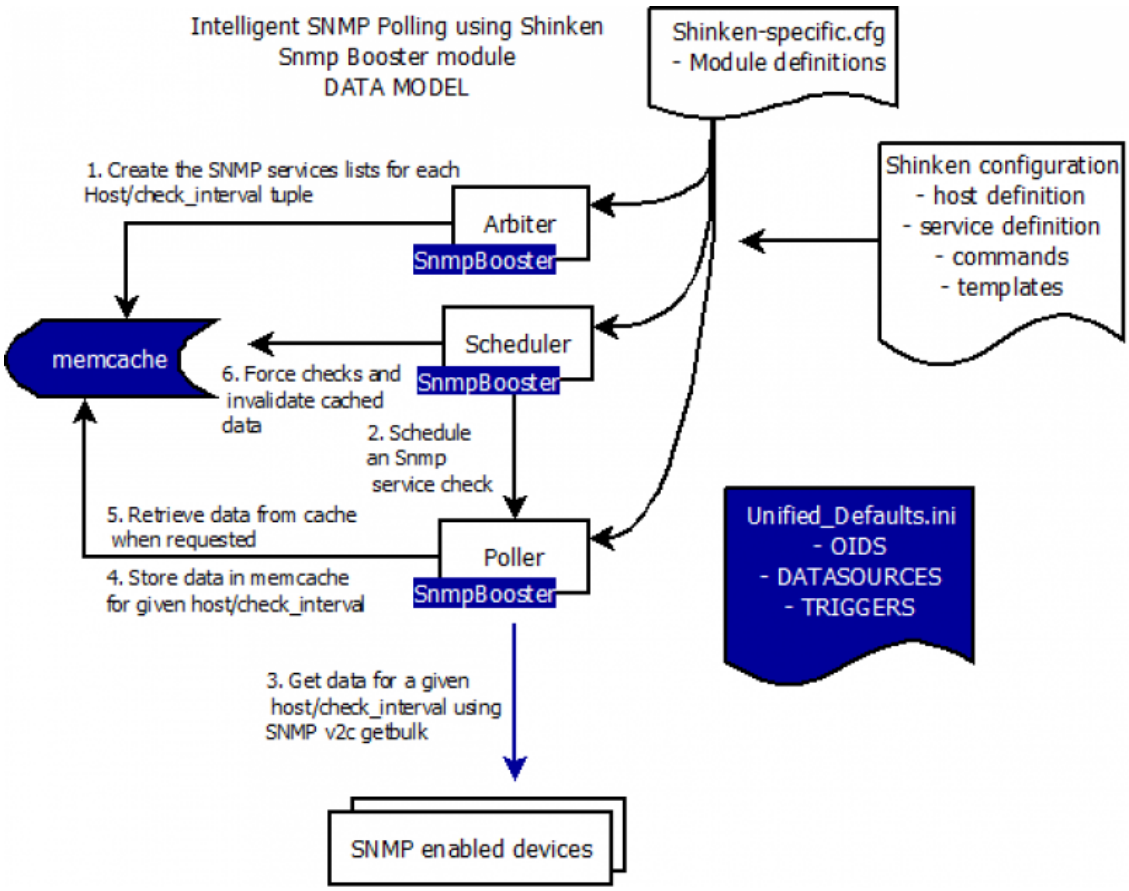

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

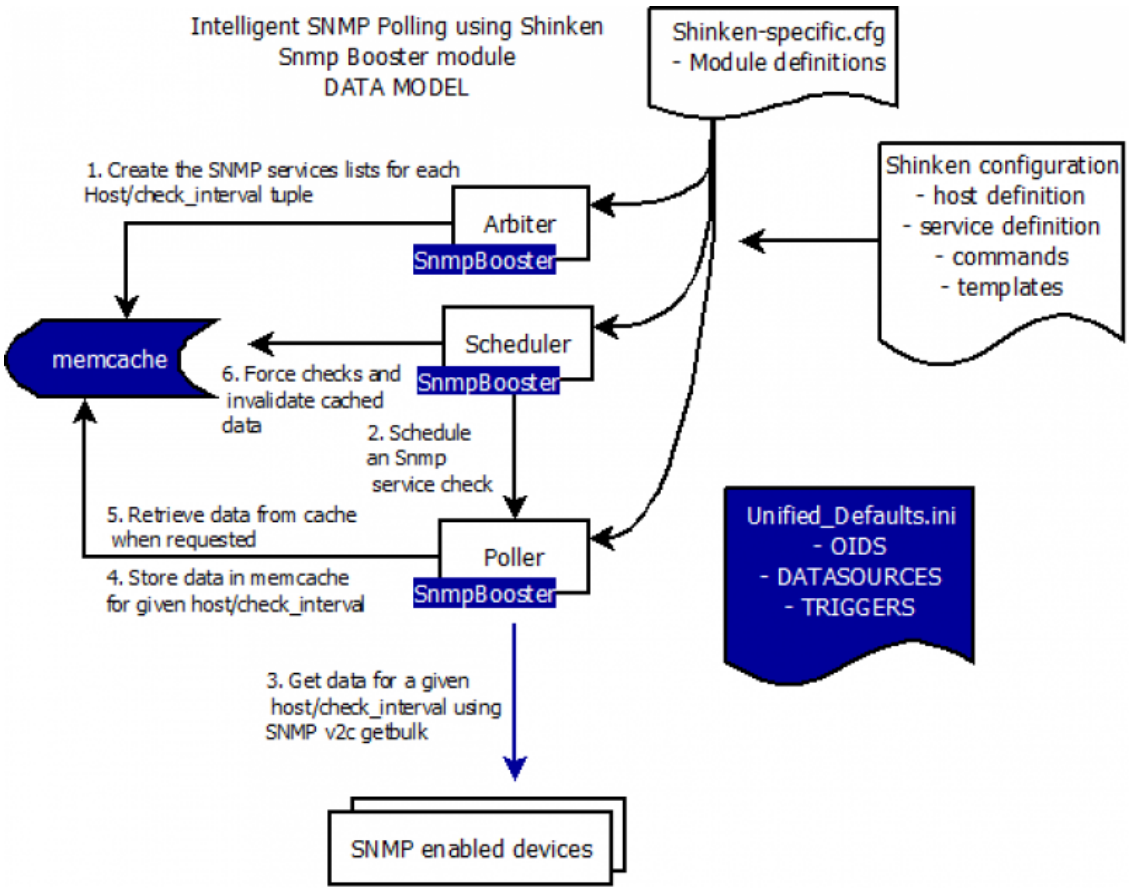

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

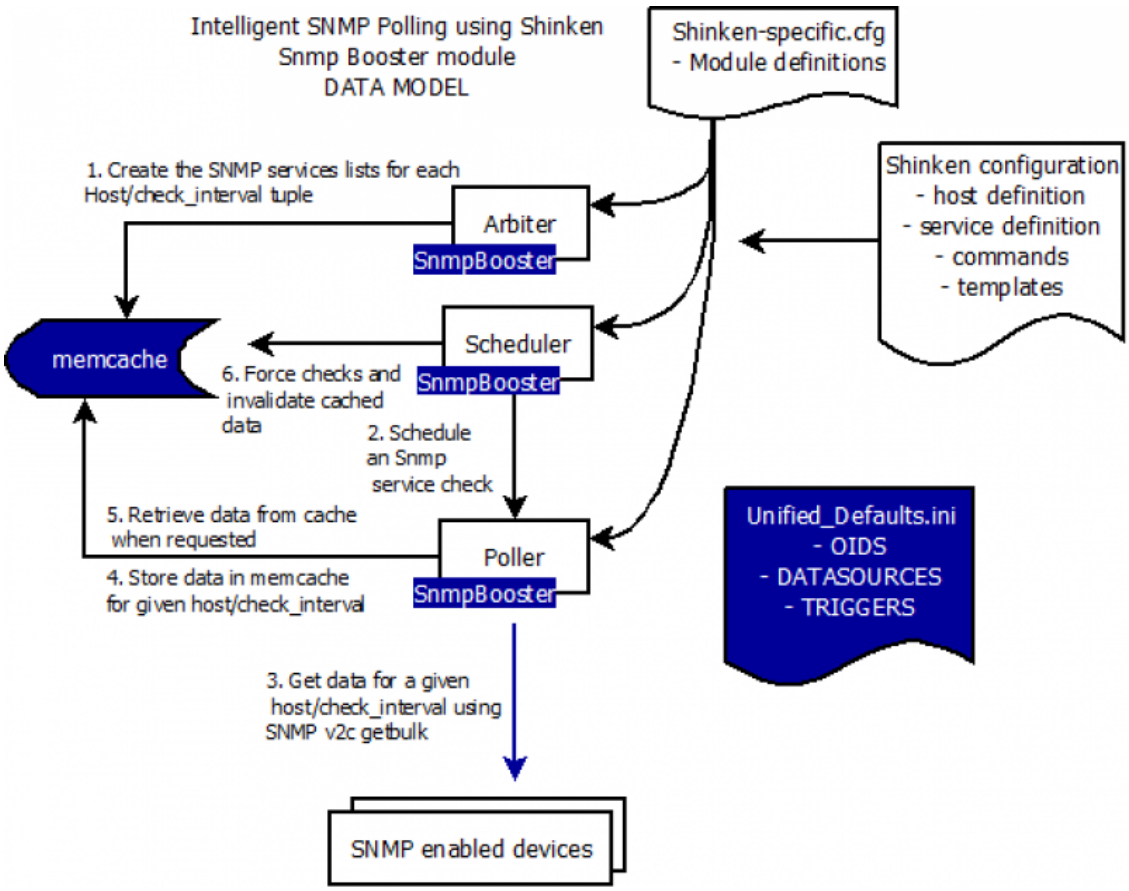

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

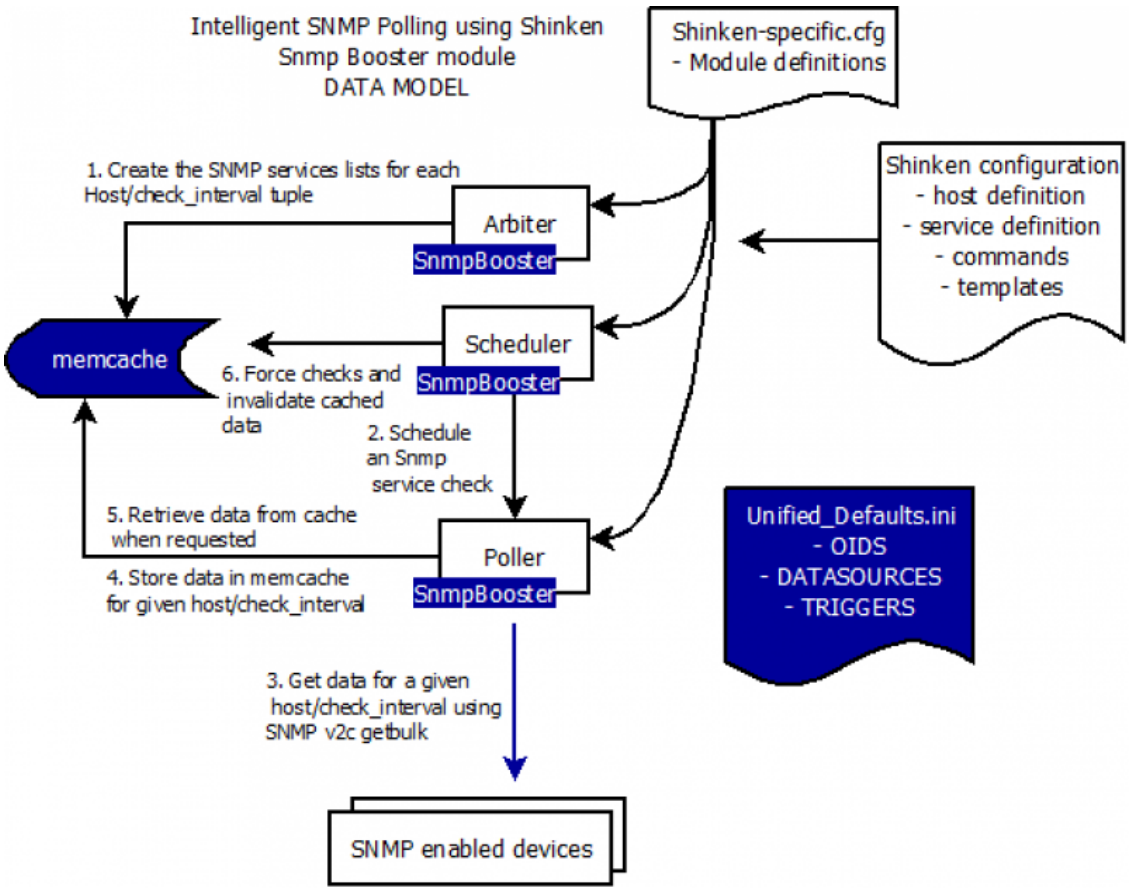

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **'Glances'_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

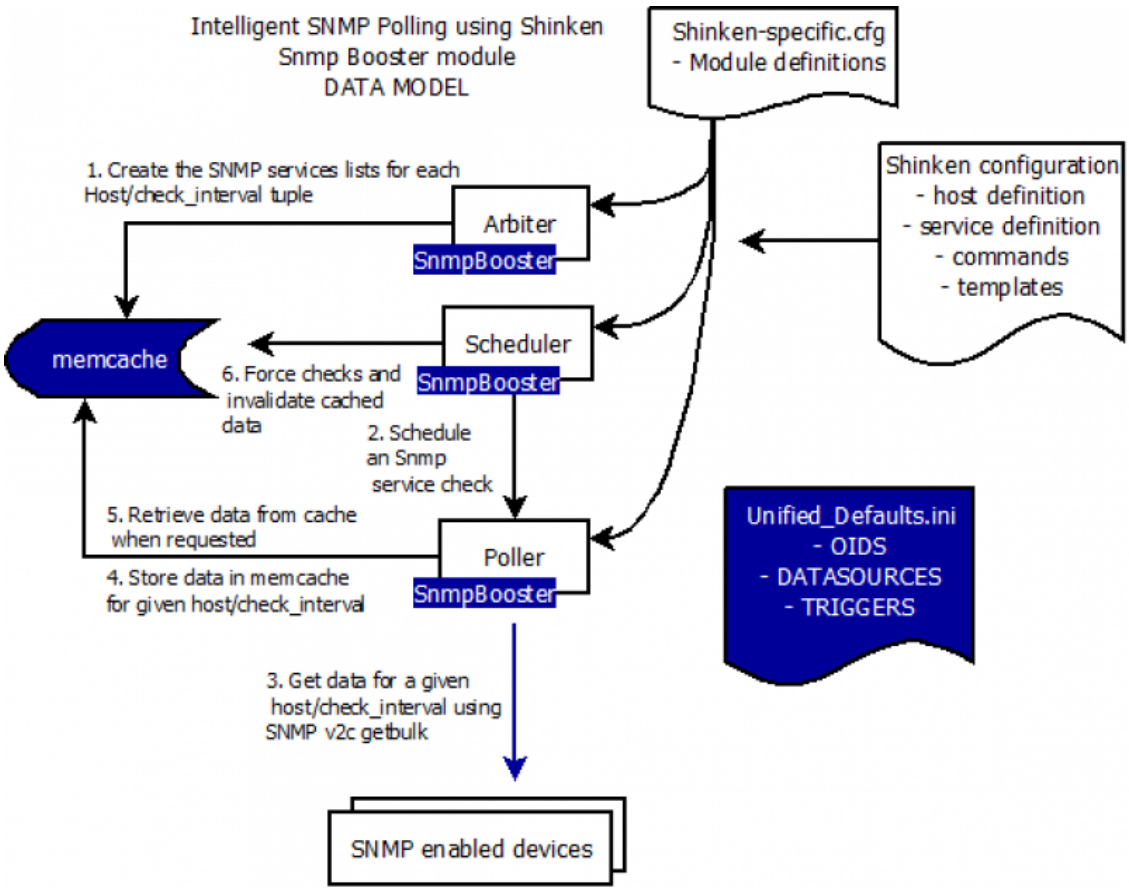

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator ” “(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py **‘Glances‘_** is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : checkglances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis

- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken

- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers

- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.

- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor) TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

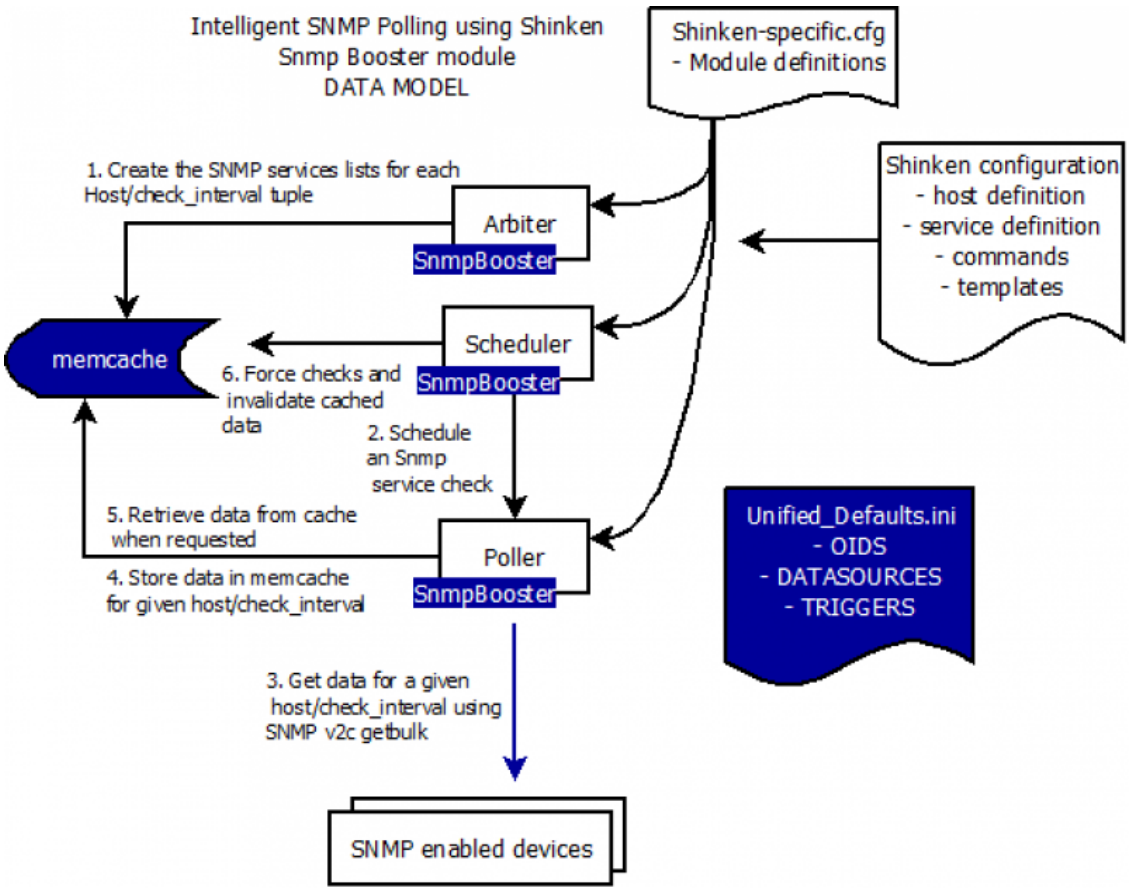

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules

The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
```

```
    port     7768                  ; tcp port of the daemon

    #optionnal: modules for things as retention
    modules    RedisRetention
    }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
    broker_name      broker-1
[...]
```

```
    modules          Simple-log,Livestatus
}
```

Disable human readable logs In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting

Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py

'Glances'_ is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : check-glances.py.

How does it work Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ? For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration

Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address? We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis
- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module

The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
: :
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting

The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

NRPE Module

What is it The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

Extending Shinken

Introduction There are a lot of "Modules" and "Addons" that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken
- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules

The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases

- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk

- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)
- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
```

```
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2 Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
     module_name      Service-Perfdata
     module_type      service_perfdata
     path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
     module_name      Host-Perfdata
     module_type      host_perfdata
     path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
     module_name              Status-Dat
     module_type              status_dat
     status_file              /usr/local/shinken/var/status.data
     object_cache_file        /usr/local/shinken/var/objects.cache
     status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor)

TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ? Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module

The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module

This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS

- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage

- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]

- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

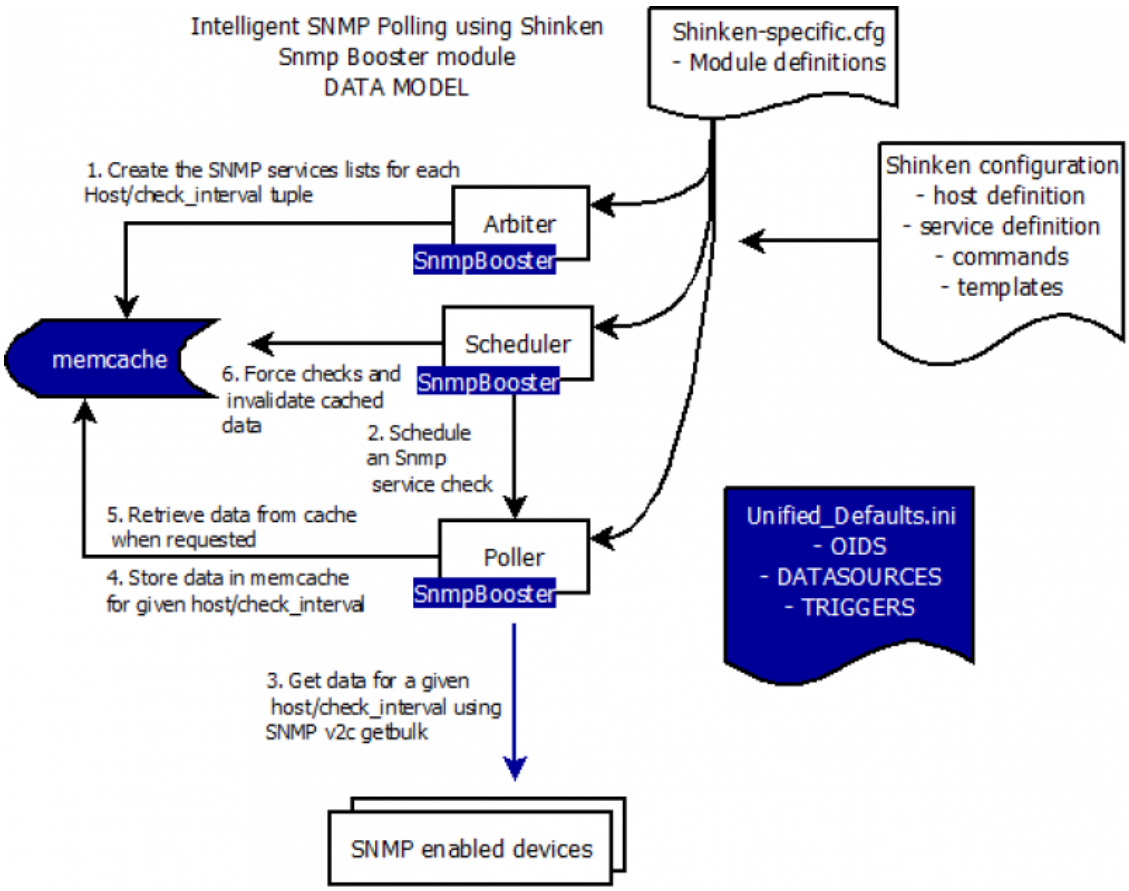

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module

This part must be splitted and sent in all module repositories

WebUI integration with Shinken The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all

- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map

- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

Amazon AWS/EC2 import

Description

You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module

In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
```

```
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts

The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

Amazon AWS/EC2 import

Description

You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module

In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts

The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
```

```
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

The distributed retention modules

The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules

If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB

MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached

Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon
```

```
     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis

Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    RedisRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

How to enable and use Livestatus module

Define the Livestatus module

Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module

Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
     broker_name      broker-1
[...]
```

```
     modules          Simple-log,Livestatus
}
```

Disable human readable logs

In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

Exporting data for reporting

Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module

The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data

Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk

Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format

The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability

The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

Monitoring Linux System with Glances and checkglances.py

'Glances'_ is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : check-glances.py.

How does it work

Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances

See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py

Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack

Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ?

For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

Shinken GLPI integration

Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module

In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

Ip Tag module

Why tag a host based on it's IP address?

We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module

The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it

In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
    module_name   IpTag
    module_type   ip_tag
    ip_range      192.168.0.1/24
    property      use
    value         dmz

    # Optional method
    method append
}
```

For a host defined as:

```
define host{
```

```
    host_name   example_host1
    address     192.168.0.2
    use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
    host_name   example_host1
    address     192.168.0.2
    use         linux,dmz
}
```

Name resolution Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

Ubuntu Landscape import

Description

You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module

In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it? All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work

Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis
- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

NSCA module

The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

Retention troubleshooting

The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ''service shinken stop''
- Save the pickle: ''pickle.dump(r, open('/tmp/retention.dat', 'w'))''
- Start Shinken: ''service shinken start''

NRPE Module

What is it

The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it

The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That's it. From now on all checks that use this command will use Shinken's NRPE module and will be launched by it.

Extending Shinken

Introduction

There are a lot of "Modules" and "Addons" that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken
- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts

- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

Broker modules

The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends

There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)

- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability.

The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing

Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing

PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2

Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Service-Perfdata
    module_type      service_perfdata
    path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
    module_name      Host-Perfdata
    module_type      host_perfdata
    path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell

The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
    module_name              Status-Dat
    module_type              status_dat
    status_file              /usr/local/shinken/var/status.data
    object_cache_file        /usr/local/shinken/var/objects.cache
    status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database This interface is no longer supported and is going to be completely removed from Shinken.

TSCA (Thrift Service Check Acceptor)

TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ?

Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation

In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

VMWare Arbiter module

How it works

The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it

The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]*%%*

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

Web Service Module

The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module

If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module

The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

SNMP module

This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS
- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator

- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage
- [xxxx] Unit tests integrated with Shinken test suite

- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]
- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

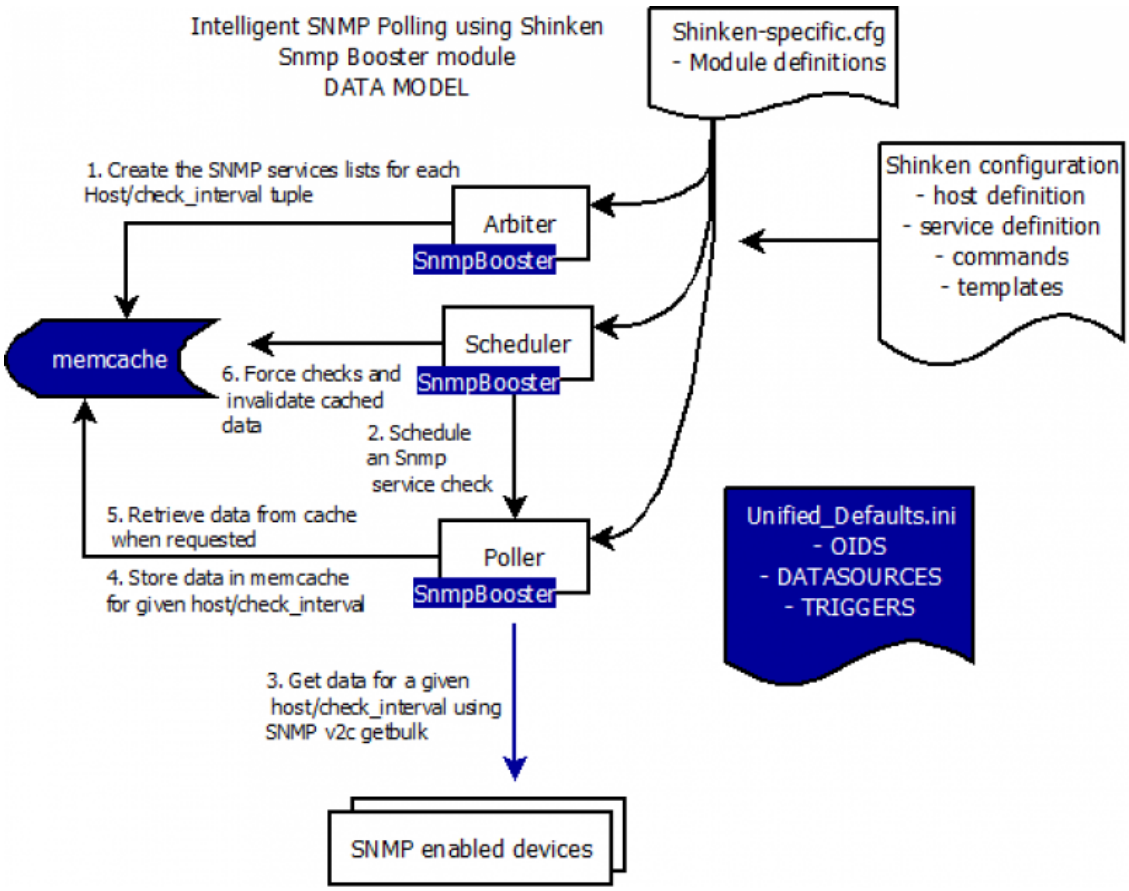

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs*

*instead of 375 request PDUs**. Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification *SnmpBooster design specification* and current development status.

Data model The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration *SnmpBooster installation*

Reference Dictionnary *SnmpBooster reference dictionary*

Troubleshooting *SnmpBooster troubleshooting*

Graph templates These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.

- Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
- Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes

On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

WebUI module

This part must be splitted and sent in all module repositories

WebUI integration with Shinken

The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User? A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page? All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.

- refresh: boolean, set the refresh to 60s or do not refresh at all
- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using? It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part
- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis

- add bp in nagvis auto-map
- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax

work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification

Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

16.1.2 Amazon AWS/EC2 import

Description

You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

Configuring the Landscape import module

In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts

The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

16.1.3 Amazon AWS/EC2 import

Description

You can import data from Amazon AWS/EC3 **'AWS'_** into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

Prerequisites

You will need the **'libcloud'_** package installed on your Arbiter server.

Configuring the Landscape import module

In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

Configuring the Arbiter module

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

Generated hosts

The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

16.1.4 The distributed retention modules

The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

Non distributed retention modules

If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

MongoDB

MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Memcached

Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

Redis

Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    RedisRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

16.1.5 How to enable and use Livestatus module

Define the Livestatus module

Start by reviewing the ''/etc/shinken/shinken-specific.cfg" file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means "listen on all interfaces".
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is "var/livelogs.db"
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

Enable the Livestatus module

Still in ''/etc/shinken/shinken-specific.cfg'', find the object Broker and add "Livestatus" to its "modules":

```
define broker{
```

```
     broker_name      broker-1
[...]
```

```
     modules          Simple-log,Livestatus
}
```

Disable human readable logs

In ''/etc/shinken/nagios.cfg'', verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

16.1.6 Exporting data for reporting

Shinken can export necessary data for reporting in 3rd party systems.

raw_socket module

The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

Why export reporting data

Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

Reporting module using Splunk

Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

Raw_socket log format

The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

Raw_socket module availability

The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

16.1.7 Monitoring Linux System with Glances and checkglances.py

'Glances'_ is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : check-glances.py.

How does it work

Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

Install glances

See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

Install checkglances.py

Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

Install the glances pack

Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

What will be monitored ?

For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

16.1.8 Shinken GLPI integration

Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

Enabling GLPI Shinken module

In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

16.1.9 Ip Tag module

Why tag a host based on it's IP address?

We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

Ip_tag module

The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

Add value instead of setting it

In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
```

```
    property      use
    value         dmz

    # Optional method
    method append
}
```

For a host defined as:

```
define host{
```

```
    host_name   example_host1
    address     192.168.0.2
    use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
    host_name   example_host1
    address     192.168.0.2
    use         linux,dmz
}
```

Name resolution

Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

16.1.10 Ubuntu Landscape import

Description

You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

Prerequisites

You will need the landscape-api package installed on your Arbiter server.

Configuring the Landscape import module

In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

Configuring the Arbiter module

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                  ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

16.1.11 Shinken Livestatus API

Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it

Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it

It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it?

All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

How does it work

Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis
- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts

- Debug scripts

Installing Livestatus

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus

Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

16.1.12 NSCA module

The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

16.1.13 Retention troubleshooting

The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ‘’service shinken stop’‘
- Save the pickle: ‘’pickle.dump(r, open(‘/tmp/retention.dat’, ‘w’))’‘
- Start Shinken: ‘’service shinken start’‘

16.1.14 NRPE Module

What is it

The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

How to define it

The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That’s it. From now on all checks that use this command will use Shinken’s NRPE module and will be launched by it.

16.1.15 Extending Shinken

Introduction

There are a lot of “Modules” and “Addons” that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken
- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net
- NagiosExchange.org
- Shinken community exchange for monitoring packs

16.1.16 Broker modules

The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

Core modules

simple_log: get all logs into one file

You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server

Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk

Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

Livestatus API Module - Thruk, Nagvis and Multisite Web frontends

There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk

- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite

Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB

Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null

In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

Network Based Modules - Graphite graphing

Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite

Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)
- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

SQL Based Modules - PNP4Nagios Graphing

PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface

Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod
- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

SQL Based Modules - Frontend Centreon2

Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database

Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt

- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
     module_name      Service-Perfdata
     module_type      service_perfdata
     path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
     module_name      Host-Perfdata
     module_type      host_perfdata
     path             /usr/local/shinken/src/var/host-perfdata
}
```

Legacy File based Modules - Old CGI or VShell

The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file

Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
     module_name              Status-Dat
     module_type              status_dat
     status_file              /usr/local/shinken/var/status.data
     object_cache_file        /usr/local/shinken/var/objects.cache
     status_update_interval   15 ; update status.dat every 15s
}
```

Deprecated or unsupported modules

SQL Based MerlinDB

This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle

This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database

This interface is no longer supported and is going to be completely removed from Shinken.

16.1.17 TSCA (Thrift Service Check Acceptor)

TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

What is thrift ?

Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

Installation

In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side

Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients

There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

16.1.18 VMWare Arbiter module

How it works

The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

How to define it

The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

16.1.19 Web Service Module

The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

Configuring the Web Service Module

If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

Using the Web Service Module

The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

16.1.20 SNMP module

This part must be splitted and sent in all module repositories

Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements

The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation

SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
```

```
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
      ds_oid = $OidmyOidDefinition
```

```
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

Shinken integrated SNMP Module

Overview

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.
- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation

- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS
- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext
- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code

- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage
- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]
- [TRIGGERGROUP]

DATASOURCE DICTIONARY OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY [DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration

This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

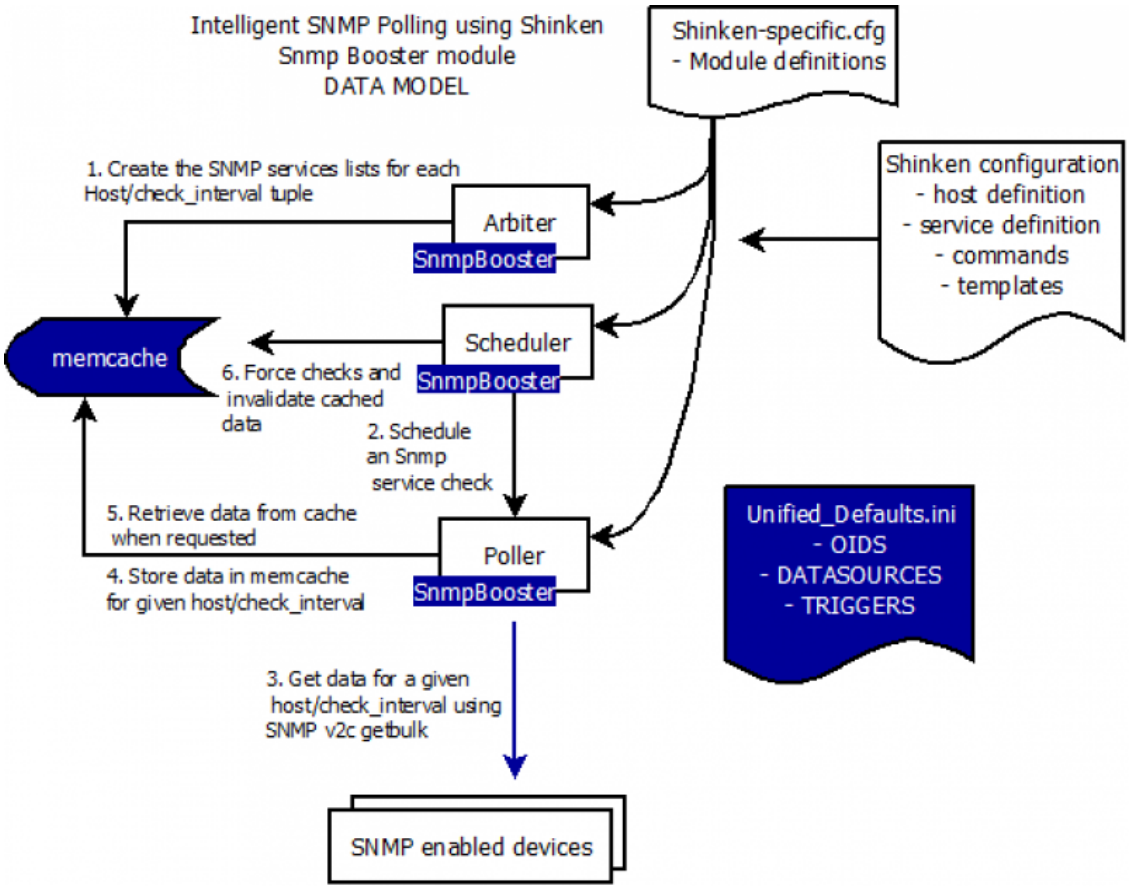

Shinken Integration

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs instead of 375 request PDUs**.* Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations

You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification

SnmpBooster design specification and current development status.

Data model

The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys
  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition

  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

  - genDevConfig plugins create Shinken configurations

Installation and configuration

SnmpBooster installation

Reference Dictionnary

SnmpBooster reference dictionary

Troubleshooting

SnmpBooster troubleshooting

Graph templates

These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency

Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements

Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments

Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.
  - Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
  - Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance

Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template

A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files

All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence

If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file

Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors__**

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors__**

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors__**

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

Memcached Notes

On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

16.1.21 WebUI module

This part must be splitted and sent in all module repositories

WebUI integration with Shinken

The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User?

A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page?

All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates

Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all
- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates

Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using?

It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part

- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map
- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

WebUI search syntax

work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

WebUI SSL Certification

Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

16.2 Amazon AWS/EC2 import

16.2.1 Description

You can import data from Amazon AWS/EC2 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC2. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

16.2.2 Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

16.2.3 Configuring the Landscape import module

In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

16.2.4 Configuring the Arbiter module

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           AWS
     }
```

Restart your Arbiter and it's done :)

16.2.5 Generated hosts

The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
```

```
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

16.3 Amazon AWS/EC2 import

16.3.1 Description

You can import data from Amazon AWS/EC3 **'AWS'**_ into Shinken to create hosts.

Amazon provide VM hosting with EC3. With this arbiter module, you will be able to load your EC2 hosts into Shinken.

16.3.2 Prerequisites

You will need the **'libcloud'**_ package installed on your Arbiter server.

16.3.3 Configuring the Landscape import module

In your shinken-specific.cfg file, just add (or uncomment):

```
define module {
 module_name      AWS
 module_type      aws_import

 # Configure your REAL key and secret for AWS
 api_key              PAAAB2CILT80I0ZA0999
 secret           GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH
 default_template generic-host        ; if the host is not tagged, use this one
 }
```

- Put in key and secret your private Landscape access. * default_template will be used if your host is not "tagged" in Landscape

16.3.4 Configuring the Arbiter module

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master

     address          localhost                   ;IP or DNS adress
     port             7770
```

```
    spare            0
    modules           AWS
    }
```

Restart your Arbiter and it's done :)

16.3.5 Generated hosts

The configuration generated will look as below :

```
define host {
    host_name            i-3fc56e5a
    address              8.8.4.4
    use                  t1.micro,MyTag,EC2,generic-host
    _EC2_AVAILABILITY    us-east-1a
    _EC2_CLIENTTOKEN
    _EC2_DNS_NAME
    _EC2_GROUPS          quicklaunch-1
    _EC2_IMAGEID         ami-1b814f72
    _EC2_INSTANCEID      i-3fc56e5a
    _EC2_INSTANCETYPE    t1.micro
    _EC2_KERNELID        aki-825ea7eb
    _EC2_KEYNAME         testaws
    _EC2_LAUNCHDATETIME  2012-09-26T12:19:38.000Z
    _EC2_LAUNCHINDEX     0
    _EC2_PRIVATE_DNS
    _EC2_PRIVATE_IP
    _EC2_PRODUCTCODE
    _EC2_PUBLIC_IP       8.8.4.4
    _EC2_RAMDISKID       None
    _EC2_STATUS          stopped
    _EC2_TAGS            demo:myvalue,use:MyTag
 }
```

Here with a stopped t1.micro instance with no name. You can put your how "use" parameter by adding a EC2 tag "use" on your VM. It will be output on the host configuration so you can setup the monitoring as you want. .. _libcloud: http://libcloud.apache.org/index.html .. _AWS: https://console.aws.amazon.com

16.4 The distributed retention modules

The high availability allow the Arbiter to send a configuration to a spare scheduler, but a spare scheduler does not have any saved states for hosts and services. It will have to recheck them all. It's better to use a distributed retention module so spares will have all the information they need to start with an accurate picture of the current states and scheduling :)

16.4.1 Non distributed retention modules

If you are just running tests on a single server, use the pickle or memcache retention modules You can also use MongoDB if you already have it installed for use with the WebUI.

16.4.2 MongoDB

MongoDB is a scalable, high-performance, open source NoSQL database written in C++.

Step 1: Install MongoDB:

We will use mongodb package from 10gen repository, so we start by adding it in apt sources list:

```
echo 'deb http://downloads-distro.mongodb.org/repo/debian-sysvinit dist 10gen' > /etc/apt/sources.lis
apt-get update
```

And we install it:

```
apt-get install mongodb-10gen
```

And we install pymongo, the python interface to MongoDB, with easy_install to avoid packages versions problems (like here **'forums.monitoring-fr.org/index.php?topic=5205.0'_**):

```
apt-get remove python-pymongo
easy_install pymongo
```

And finally we start MongoDB :

```
/etc/init.d/mongodb start
```

Step 2: define a mongodb_retention module in your shinken-specific.cfg file:

```
define module {
  module_name MongodbRetention
  module_type mongodb_retention
  uri mongodb://localhost/?safe=true
  database shinken
}
```

Step 3: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    MongodbRetention
     }
```

Step 4: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

16.4.3 Memcached

Memcached is a distributed memory resident key/value server. It's very easy to install:

```
sudo apt-get install memcached
sudo /etc/init.d/memcached start
```

The shinken module also needs the python-memcache package to talk to this server.

```
sudo apt-get install python-memcache
```

To use it,

Step 1: define a memcache_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      MemcacheRetention
     module_type      memcache_retention
     server           127.0.0.1
     port             11211
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optional: modules for things as retention
     modules    MemcacheRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :)

16.4.4 Redis

Redis is a distributed key/value server (on disk and in-memory). It's very easy to install:

```
sudo apt-get install redis-server
sudo /etc/init.d/redis-server start
```

The shinken module also need the python-redis package to talk to this server.

```
sudo apt-get install python-redis
```

Step 1: define a redis_retention module in your shinken-specific.cfg file:

```
define module{
     module_name      RedisRetention
     module_type      redis_retention
     server           127.0.0.1
}
```

Step 2: Declare a retention module for your scheduler(s) :)

Example:

```
#The scheduler is a "Host manager". It get hosts and theirs
#services. It scheduler checks for them.
define scheduler{
     scheduler_name   scheduler-1   ; just the name
     address  localhost             ; ip or dns address of the daemon
     port     7768                  ; tcp port of the daemon

     #optionnal: modules for things as retention
     modules    RedisRetention
     }
```

Step 3: Restart the Arbiter, and your Scheduler will now save its state between restarts. :) .. _forums.monitoring-fr.org/index.php?topic=5205.0: http://forums.monitoring-fr.org/index.php?topic=5205.0

16.5 How to enable and use Livestatus module

16.5.1 Define the Livestatus module

Start by reviewing the ‘’/etc/shinken/shinken-specific.cfg” file for this module:

```
define module{
```

```
module_name      Livestatus
module_type      livestatus
host             *       ; * = listen on all configured IP addresses
port             50000   ; port to listen on
database_file    /var/lib/shinken/livestatus.db
```

}

With these parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is *, which means “listen on all interfaces”.
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is “var/livelogs.db”
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these **must be IP-addresses, NOT host names**. Because DNS lookups for every incoming livestatus request could hang up and therefore block the module.

16.5.2 Enable the Livestatus module

Still in ‘’/etc/shinken/shinken-specific.cfg’‘, find the object Broker and add “Livestatus” to its “modules”:

```
define broker{
```

```
     broker_name      broker-1
[...]
```

```
     modules          Simple-log,Livestatus
}
```

16.5.3 Disable human readable logs

In ‘’/etc/shinken/nagios.cfg’‘, verify that the option **human_timestamp_log** is set to **0**.

In version 0.6.5, you can not have at the same time a simple-log file with human readable timestamp and a Livestatus database.

16.6 Exporting data for reporting

Shinken can export necessary data for reporting in 3rd party systems.

16.6.1 raw_socket module

The raw socket module exports reporting data using a syslog format to a raw TCP socket. Theses logs can be captured, parsed and indexed by any log management system like Splunk or Logstash.

16.6.2 Why export reporting data

Custom UIs like Thruk, Multisite, WebUI, Adagios, Nagios IX, etc. All have poor or inflexible reporting. Simply because these are not reporting engines.

This also permits to have a generic reporting method that can accept data from any Nagios, Shinken, Icinga or even Riemann.

16.6.3 Reporting module using Splunk

Splunk was selected as the reporting platform for creating, managing and presenting reports. Though the raw_socket module does not have any Splunk specific particularities. It is notable that Splunk reports can be embedded in any 3rd party UI using its REST API.

16.6.4 Raw_socket log format

The logs are plain-text and compliant with RFC3164. They use an intelligent date format, a configurable seperator " "(space by default), and key=value pairs in the message data portion of the log. Should a key or value contain a space they must be double quoted. This is syslog using structured data without going to full out JSON that may not be compatible with all log management systems.

The message fields that are supplied in each status update in addition to the standard syslog header are:

- Hostname
- Servicename
- Business impact
- Host/Service status
- Host/Service previous status
- Acknowedgement status
- Maintenance status

Any time the maintenance, host, service or acknowledgement status changes, a log is sent by the Shinken raw_socket module.

Having all relevant data in each log makes it very easy to compute reports for given time periods.

The hostgroup or service group memberships, custom variables or any other configuration related data should be queried from the monitoring system using its API (Livestatus in the case of Shinken and Nagios).

16.6.5 Raw_socket module availability

The module is available in both Shinken 1.4 and Shinken 2.0. The module will be integrated in master branches once unit tests and end-to-end testing is completed. 3rd quarter 2013.

16.7 Monitoring Linux System with Glances and checkglances.py

'Glances'_ is a fantastic multiplatform system monitoring tool writen by Nicolargo. Glances can run on (Linux, Windows, MacOS ...). There is a growing ecosystem around Glances and of course a Nagios/Shinken plugin : check-glances.py.

16.7.1 How does it work

Glances can run in server mode. It provide a XML-RPC server (over http) and allow to retrieve system information throuh a restful API. The only thing to do is to install and start Glances as server on the monitored host, then install checkglances.py on the shinken server and finaly install the glances pack.

16.7.2 Install glances

See the glances documentation to install glances on Linux.

<code bash> easy_install glances </code>

There is a debian startup script in the glances repository. And another one for centos here.

16.7.3 Install checkglances.py

Just follow the documentation for the checkglances.py prerequisites and put it in your nagios plugins folder.

16.7.4 Install the glances pack

Open a terminal as the shinken user and just type the following command

<code bash> shinken install glances </code>

Then just edit the configuration for a linux monitored host and tag it as **glances**

```
define host{
 name mylinuxhost
 use glances
 ...
```

Finaly restart the shinken arbiter

```
service shinken-arbiter restart
```

16.7.5 What will be monitored ?

For the moment just the basic system (ram, cpu, load, memory, filesystem and network).

If you want to modify the default thresholds for just a host you can do it by overiding the thresholds macro in your host definition. If you want to do it for several hosts, declare a new template and just add the desired thresholds macro to be overide.

Here is a list of the macros

```
::
```

> _LOAD_WARN 2 _LOAD_CRIT 3 _CPU_WARN 80 _CPU_CRIT 90 _MEMORY_WARN 90 _MEMORY_CRIT 95 _FS /, /home _FS_WARN 90 _FS_CRIT 95 _IFACES eth0, eth1 _NET_WARN 7500000 _NET_CRIT 10000000

Note the _FS and _IFACES macros. They leverage the ability to duplicate service for a given list of things to be monitored. Here you see that / and /home partitions will be monitored. If you want to modify this then just add your own in a new template that will inherit the glances template.

16.8 Shinken GLPI integration

Shinken supports importing hosts from GLPI

For people not familiar with GLPI, it is an Open-Source CMDB. Applicable to servers, routers, printers or anything you want for that matter. It is also a help-desk tool. GLPI also integrates with tools like FusionInventory for IT inventory management.

You can still have flat files AND GLPI if you want. Hosts imported from GLPI will be treated as standard hosts from flat file (inheritance, groups, etc). In fact, in this first version the module only sends the host_name to Shinken, but there should be more information extracted like network connexions for parents relations in the future. :)

16.8.1 Requirements

- python module MySQLdb
- Compatible version of GLPI Shinken module and GLPI version - ...
- etc..

16.8.2 Enabling GLPI Shinken module

In your shinken-specific.cfg file, just add (or uncomment):

```
#You know GLPI? You can load all configuration from this app(
#with the webservices plugins for GLPI, in xmlrpc mode
#and with plugin monitoring for GLPI)
# =============== Work with Plugin Monitoring of GLPI ===============
#All configuration read from this will be added to the others of the
#standard flat file
define module{
     module_name      GLPI
     module_type      glpi
     uri              http://localhost/glpi/plugins/webservices/xmlrpc.php
     login_name       glpi
```

```
     login_password   glpi
}
```

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           GLPI
     }
```

It's done :)

16.9 Ip Tag module

16.9.1 Why tag a host based on it's IP address?

We learned on the *DMZ monitoring* that you can "tag" a host with a poller_tag so it will be checked only by DMZ pollers. It can be tedious process to manually tag all these hosts. Shinken provides an automated method to do this.

This is handled by the arbiter ip_tag module.

16.9.2 Ip_tag module

The ip_tag module is an arbiter module that looks up the host address during the configuration parsing and compares it to an predefined IP range (IPV4 or IPV6). If the range matches, it can apply or add a property, such as poller_tag. Other types of tags could be related to locations, functions, administrative groups, etc.

Method: replace or append.

replace will put the value if the property is unset. This is the default method. .. warning:: Fix me: replace, implies overriding an existing value. And will cause user issues down the line. Rename it to set. Replace should be a different method altogether, so overall there should be 3 methods(set, append, replace).

append will add with a , if a value already exist

Here is an example that will set the poller_tag property with the value DMZ of all hosts that:

- have a their hostadress ip (or name resolved) in the range 192.168.0.1/24.
- do not already have a poller_tag property set

```
define module{
```

```
     module_name     IpTag
     module_type     ip_tag
     ip_range        192.168.0.1/24
     property        poller_tag
     value           DMZ
}
```

To use the module add it in the arbiter object:

```
::
```

define arbiter { [...] modules NSCA,IpTag

}

16.9.3 Add value instead of setting it

In the previous example we set the property poller_tag to a value. But what if we want to "add" a value, like a template name? We can change the "method" property of the module to "append" so we can add the "value" to the host property instead of replace it.

Here is an example where we want to add the "dmz" template on the hosts, while also keeping any defined templates:

```
define module{
```

```
     module_name   IpTag
     module_type   ip_tag
     ip_range      192.168.0.1/24
     property      use
     value         dmz

     # Optional method
     method append
}
```

For a host defined as:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux
}
```

It will be "changed" internaly in:

```
define host{
```

```
     host_name   example_host1
     address     192.168.0.2
     use         linux,dmz
}
```

Name resolution

Of course you don't need to set IP address in the address of your host. You can set a hostname, and it will be resolved, and then compared to the range. :)

16.10 Ubuntu Landscape import

16.10.1 Description

You can import data from Ubuntu Landscape into Shinken to create hosts.

Ubuntu Landscape, is a web app to manage all your linux servers (update and co), which is provided by Canonical, Ubuntu's parent company.

This is an Arbiter import module. The Arbiter manages the configuration.

16.10.2 Prerequisites

You will need the landscape-api package installed on your Arbiter server.

16.10.3 Configuring the Landscape import module

In your shinken-specific.cfg file, just add (or uncomment):

define module { module_name Landscape module_type landscape_import

Configure your REAL key and secret from Landscape key PAAAB2CILT80I0ZA0999 secret GGtWAAAzEItz0utWUeCe9BJKIYWX/hdSbA6YCHHH default_template generic-host ; if the host is not tagged, use this one

If you are using a specific certificate for landscape_api #ca /path/to.ca.cert

If you need a proxy to access https://landscape.canonical.com/api/ #https_proxy http://user:secret@myproxy.com:3128

}

- Put in key and secret your private Landscape access.
- default_template will be used if your host is not "tagged" in Landscape
- If you are using a specific certificate for connexion, give the full path in ca
- You will certainly need an https proxy to access the Landscape API. If so, configure it there.

16.10.4 Configuring the Arbiter module

And add it in your Arbiter object as a module.

```
define arbiter{
     arbiter_name     Arbiter-Master
#    host_name       node1       ;result of the hostname command under Unix
     address          localhost                   ;IP or DNS adress
     port             7770
     spare            0
     modules           Landscape
     }
```

Restart your Arbiter and it's done :)

Tip: Your "tags" will be applied as templates for your hosts.

16.11 Shinken Livestatus API

16.11.1 Overview

- Jump back to Getting Started
- Jump back to *Livestatus module configuration*

What is it

Livestatus API is the modern method of interacting with Shinken and Nagios based systems alike.

Originally developed for Nagios, MK Livetstatus, was re-implemented in Python for use with Shinken by professional developers. The access methods and query languages are the same.

Why use it

It is fast.

It is network based using TCP.

It is standard for all Nagios and Shinken monitoring systems.

It uses in-memory structures for status data

It uses a back-end database for historical reporting

It uses an SQL based query language

It has a host of powerful nuts and bolts features (authentication, caching, connection persistence, multiple object methods (JSON, Python)).

What can I do with it?

All programs and scripts use (or should use) this method to :

- Obtain the current state of hosts and services
- Obtain the current configuration related to hosts and services
- Send administrative commands (downtimes, comments, forcing checks)
- Filter information to display
- Obtain basic reports and event statistics

Shinken being a modern re-implementation of Nagios has additional features that are available in addition to the standard Livestatus API.

Shinken Livestatus API unique features can :

- Obtain Business Impact information
- Obtain intelligent dependency status hosts or services (combines dependency and parent information)
- Obtain Business Process states (same syntax as standard host and service queries)

16.11.2 How does it work

Livestatus uses a powerful SQL-like query language. The standard methods and parameters are defined at :

'MK Livestatus reference methods and query language definition'_

Full list of Shinken Livestatus API methods:

'Shinken mapping.py methods'_

16.11.3 What software uses Livestatus to interact with Shinken

Well known Web frontends

- MK Multisite
- Thruk
- NagVis
- Splunk for Nagios

Mobile Web frontends

- get list

Scripts

- Downtime management scripts
- Debug scripts

Installing Livestatus

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the *10 minute installation guide* for the *requirement lists*.

How to enable Livestatus

Very simple, jump to the *Shinken Broker configuration* section to enable it. .. _Shinken mapping.py methods: https://github.com/shinken-monitoring/mod-livestatus/blob/master/module/mapping.py .. _MK Livestatus reference methods and query language definition: http://mathias-kettner.de/checkmk_livestatus.html

16.12 NSCA module

The NSCA daemon module is used to receive send_nsca packets and submit them to the Shinken command pipe. The NSCA module can be loaded by the Receiver or Arbiter process. It will listen on port TCP/5667 for send_nsca packets.

Tip: Passive checks can be submitted *natively to the Shinken command pipe* or from remote hosts to modules, such as NSCA, AMQP or collectd, loaded in the Shinken Arbiter or Receiver process. AMQP is implemented for integration with the Canopsis Hypervisor, but could be re-used for generic messaging.

Note: The Shinken NSCA module implementation is currently limited to the "xor" obfuscation/encryption.

In your shinken-specific.cfg file, just add (or uncomment):

```
::
```

#You can send check result to Shinken using NSCA #using send_nsca command define module{

module_name NSCA module_type nsca_server host * port 5667 encryption_method 0 password helloworld

}

define receiver{

```
receiver_name    receiver-1
address          localhost
port             7773
modules          NSCA

timeout          3             ; 'ping' timeout
data_timeout     120           ; 'data send' timeout
max_check_attempts       3     ;  if at least max_check_attempts ping failed, the node is DEAD

#advanced
realm    All
}
```

This daemon is totally optional.

It's main goal is to get all passive "things" (checks but why not other commands) in distant realms. It will act as a "passive receive buffer" and will then dispatch the data or commands directly to the appropriate Scheduler or Arbiter process.

Data can be received from any Realm, thus the Realm option is nonsensical.

For now there is no init.d script to launch it. .. note:: Verify that the init script has been added.

It is launched like all other daemons:

```
bin/shinken-receiver -c etc/receiverd.ini
```

Tip: Alternatively, for small installations with you can configure a modules inside your Arbiter instead of the Receiver. It will listen the TCP/5667 port for send_nsca packets.

To configure the NSCA module in your Arbiter instead of Receiver. Add the NSCA module to your Arbiter object configuration.

```
::
```

define arbiter{ arbiter_name Arbiter-Master

 # host_name node1 ;result of the hostname command under Unix address localhost ;IP or DNS adress port 7770 spare 0 modules NSCA

}

16.13 Retention troubleshooting

The retention file contains the state of your host and services, so that they are kept when shinken is restarted.

Sometimes you'll want to inspect (and possibly fix) the retention file, in case you need to work-around a bug or simply better understand what is stored there.

When you use the PickleRetention module in the scheduler (cf. ''define scheduler" in ''shinken-specific.cfg''), you can simply manipulate the retention file using Python.

For this you can use the following code.

- It assumes Shinken is installed in ''/opt/shinken" with the retention file in ''/tmp/retention.dat'':
- Run and play with it interactively using ''ipython -i inspect_retention.py''.

```
#!/usr/bin/python
import pickle
import sys
sys.path.append('/opt/shinken')
import shinken

r = pickle.load(file('/tmp/retention.dat'))
h = r['hosts']

# Example:
# [(k,v) for (k,v) in h['your_host'].items() if k.find('downtime') >= 0]
#
# del h['your_host']
#
# for k,v in r['services'].items():
#     if 'your_host' in k:
#         del r['services'][k]
```

To save your changes:

- Stop Shinken: ''service shinken stop''
- Save the pickle: ''pickle.dump(r, open('/tmp/retention.dat', 'w'))''
- Start Shinken: ''service shinken start''

16.14 NRPE Module

16.14.1 What is it

The NRPE module allows Shinken Pollers to bypass the launch of the check_nrpe process. It reads the check command and opens the connection by itself. It scales the use of NRPE for active supervision of servers hosting NRPE agents.

The command definitions should be identical to the check_nrpe calls.

16.14.2 How to define it

The definition is very easy (and you probably just have to uncomment it):

```
define module{
     module_name       NrpeBooster
     module_type       nrpe_poller
}
```

Then you add it to your poller object:

```
define poller {
    [...]
```

```
    modules   NrpeBooster
}
```

Then just tag all your check_nrpe commands with this module:

```
define command {
   command_name   check_nrpe
   command_line   $USER1$/check_nrpe -H $HOSTADRESS$ -c $ARG1$ -a $ARG2$
   module_type    nrpe_poller
}
```

That's it. From now on all checks that use this command will use Shinken's NRPE module and will be launched by it.

16.15 Extending Shinken

16.15.1 Introduction

There are a lot of "Modules" and "Addons" that are available for Shinken. Shinken can be extended by using different types of methods.

- Web interfaces from various open-source vendors can be used (Based on Livestatus API)
- Shinken itself can be extended using modules that are embedded into the various daemon processes

** (Arbiter, Poller, Receiver, Broker, Scheduler, Reactionner) * External software can provide functional enhancements for reporting, management or usability aspects

Shinken itself provides methods to extend the base functionality:

- *Broker modules* provide external access to runtime data from Shinken
- Poller modules such as *NRPE* and the *SNMP*
- Receiver modules like *NSCA* provide acquisition methods for Shinken
- Arbiter modules permit manipulating the configuration
- The webUI that permits embedding web interface elements

Independent software that works in concert with Shinken is available for:

- Monitoring remote hosts through agent software (*NIX, Windows, etc.)
- Submitting passive checks from remote hosts
- Managing the config files through a web interface
- Mobile phone UIs
- Desktop UIs and Hypervisors
- Desktop notification addons
- ...and much more

You can find many addons that are compatible with Shinken by visiting. The only caveat is that the addons should not use the status.dat file and preferably the modern API interfaces like Livestatus:

- Nagios.org
- SourceForge.net

- NagiosExchange.org
- Shinken community exchange for monitoring packs

16.16 Broker modules

The broker module receives messages from the Schedulers related to host, service check results. In addition to that it receives a copy of the compiled configuration at startup. What is special about the broker, is that without modules, the broker does nothing. It makes the configuration and result data available to modules. Modules are the ones in charge of doing something with the data. This could be exporting this "raw" data into flat files, databases or exposing it as an API.

The modules are presented according to their use cases.

- Core modules
- Livestatus Module for frontends
- Exporting data to metric databases
- SQL based Modules for frontends
- Exporting data to logging management systems
- Legacy modules for migration purposes
- Modules no longer supported

16.16.1 Core modules

simple_log: get all logs into one file

You can get a log for all your daemons in a single file. With the exception of Debug level logs which are not sent through the network as a safe guard. Shinken state messages are ONLY logged at the debug level in the local log files. These are the ALL CAPS states that some 3rd party interfaces use to mine data from the historical logs. Actual state messages are sent as objects between the various daemons.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: simple_log
- path: path of the log file

Here is an example:

```
define module{
```

```
     module_name      Simple-log
     module_type      simple_log
     path             /usr/local/shinken/var/nagios.log
     archive_path     /usr/local/shinken/var/archives/
}
```

Syslog - Send all logs to a local syslog server

Send all logs to a local syslog server. Support for remote syslog can be implemented if requested. Just lookup the syslog module, easy to modify. Use for example with Splunk, graylog2, Logstash, Elastic search, Kibana, or other log management and analysis system.

```
define module{
```

```
  module_name Syslog
  module_type syslog
}
```

Lightweight forwarder to Splunk

Shinken state data from the local syslog can also be sent to a Splunk server for indexing, reports, SLA and analysis. Via syslog or better yet, the Splunk lightweight forwarder which supports, TCP, buffering and security parameters.

16.16.2 Livestatus API Module - Thruk, Nagvis and Multisite Web frontends

There are three well known frontends and various small tools that can interface with the Livestatus API. This API is currently the most efficient method of sharing live data with external interfaces. It provides a rich query language similar to SQL, authentication, command execution, caching,connections persistence, multiple object methods (JSON, Python). All Livestatus data is kept in memory, there is no disk IO involved. It exposes configuration objects, the current state of the hosts and services as well as historical Shinken state log messages.

- MK Multisite
- Thruk
- NagVis

Have you installed the required packages to use the Livestatus module? You can look at the requirement section of the 10 minute installation guide for the requirement lists.

Livestatus Module

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: livestatus
- host: IP interface to listen to. The default is '127.0.0.1'. '*' means 'listen on all interfaces'.
- port: TCP port to listen to.
- socket: Unix socket to listen to.
- allowed_hosts: a comma-separated list of IP-addresses which are allowed to contact the TCP port. Please keep in mind that these must be ip-addresses, NOT host names. (DNS lookups for every incoming livestatus request could hang up and therefore block the module)
- modules: a comma-separated list of modules (see below)
- pnp_path: relevant if you use the multisite gui. pnp_path points to the perfdata directory where the pnp4nagios data reside (it's the directory which contains a subdirectory for each host). Multisite needs it to display a small perfdata icon if a rrd file exists.
- To deactivate an input, the port- and socket-attributes can also take the value "none".

The Livestatus module itself can have sub-modules. In fact, it **must** have at least one submodule responsible for storing log events to a database, SQLite or MongoDB.

Livestatus module - Logstore Sqlite

Stores log-broks to an sqlite database.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_sqlite
- use_aggressive_sql: Prefer the use of sql queries to pre-filter data prior to python filtering. This is required for very large installations, as the Python structures are not indexed when filtering. Optional argument set to 0 by default. To enable set the value to 1. Some erroneous output may occur, but informal testing has shown these corner cases do not occur.
- read_only: don't send any update to the database. Should be used when you run several livestatus modules : 1 instance is R/W, other are read only.
- database_file: the path to the sqlite database file which is used to store the log broks/messages. The default is 'var/livestatus.db'
- archive_path: the path to a subdirectory where database file archives will be created. Every day the data from the last day will be moved to an archive datafile. The default is 'var/archives'.
- max_logs_age: the maximum age of the log messages (before they are deleted from the database). The default is 1 year. The argument to this parameter takes the form <number>[<period of time>], where <period of time> can be d for days, w for weeks, m for months and y for years. This parameter is currently ignored. In future releases it will be used to automatically delete ancient database files.

Here is an example:

```
define module{
```

```
     module_name      Livestatus
     module_type      livestatus
     host             *       ; * = listen on all configured ip addresses
     port             50000   ; port to listen
     socket           /usr/local/shinken/var/rw/live
     modules          logsqlite
}
define module{
```

```
     module_name      logsqlite
     module_type      logstore_sqlite
     use_aggressive_sql     0 ; optional, by default set to 0. Only for large installations.
     database_file    /usr/local/shinken/var/livelogs.db
     max_logs_age     3m ; three months. Other time intervals are d=days, w=weeks, y=years
}
```

Livestatus module - Logstore MongoDB

Stores historical log broks(inter daemon Shinken messages) to a MongoDB database. MongoDB is a distributed and very performant database that permits resilience and high availability. The current implementation has a few known broken pieces (some search functions are broken) and there are bugs in the implementation, so it is considered experimental until power users and developers have ironed out the bugs. It is the database of the future for Shinken.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_mongodb
- mongodb_uri: The address of the master. (Default: i_cant_write_the_uri_here_it_messes_the_wiki_up)
- replica_set: If you are running a MongoDB cluster (called a "replica set" in MongoDB), you need to specify it's name here. With this option set, you can also write the mongodb_uri as a comma-separated list of host:port items. (But one is enough, it will be used as a "seed")
- database: <undocumented>
- collection: <undocumented>
- max_logs_age: <undocumented>

The configuration looks quite similar to the sqlite one. **In a single-node installation and with decent amount of log traffic, the sqlite backend should be considered best practice, as it needs no extra software and is stable and fast (when run in-memory).**

Livestatus module - Logstore Null

In case you don't need any logging (for instance, if you dedicate a livestatus module instance to nagvis), you can use this module.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: logstore_null

16.16.3 Network Based Modules - Graphite graphing

Graphite is a graphing and data analysis tool. It is composed of a web frontend (graphite-web), fixed size databases (whisper) and data redistribution/munging daemon. (carbon) The first step is *having Graphite installed somewhere*, if you do not have Graphite installed, please do this and come back later.

Pre-requisite : Shinken 1.2.2+ is recommended for the best experience with Graphite.

graphite_perfdata: exports check result performance data to Graphite

Export all performance data included in check results to Whisper databases managed by Graphite. Data is buffered, then exported once every broker time tick. (Once per second). Should a communication loss occur, it will buffer data for 5 minutes before dropping data. The tick_limit max is configurable, the only limitation is the amount of memory the buffered data can eat.

The graphite broker module (v1.2) sends data to graphite in the following format :

HOSTNAME.SERVICENAME.METRICNAME

It can be optionally enhanced to send it the following format :

_GRAPHITE_PRE.HOSTNAME.SOURCE.SERVICENAME.METRICNAME._GRAPHITE_POST

Each of the three variables are optional :

- The _GRAPHITE_PRE variable can be set per host/service to provide a more flexible naming convention. (ex. Prod.site1.Hostname.servicename)

- The _GRAPHITE_POST variable can be set per host/service to specify the more information concerning the variable being stored that graphite can use to determine the retention rule to use for the variable.
- The SOURCE variable can be set in shinken-specific.cfg for the WebUI module and the Graphite broker.

Metric names are converted from Shinken's format to a more restrictive Graphite naming. Unsupported characters are converted to "_" underscore. So do not be surprised to see that some names differ in the WebUI or directly in Graphite. Permitted characters in Graphite are "**A-Za-z_**.", the hyphen character is permitted from Shinken, though only future versions of Graphite will support it.

Performance data transfer method can be set to pickle, which is a binary format that can send data more efficiently than raw ascii.

It takes the following parameters:

- module_name: name of the module called by the brokers
- module_type: graphite_perfdata
- host: ip address of the graphite server running a carbon instance
- port: port where carbon listens for metrics (default: 2003 for raw socket, 2004 for pickle encoded data)
- use_pickle: Use a more efficient transport for sending check results to graphite instead of raw data on the socket (v1.2)
- tick_limit: Number of ticks to buffer performance data during a communication outage with Graphite. On tick limit exceeded, flush to /dev/null and start buffering again. (v1.2)
- graphite_data_source: Set the source name for all data coming from shinken sent to Graphite. This helps diferentiate between data from other sources for the same hostnames. (ex. Diamond, statsd, ganglia, shinken) (v1.2)

Here is an example:

```
define module {
```

```
module_name Graphite-Perfdata
host xx.xx.xx.xx
module_type graphite_perfdata
port 2004 ; default port 2003 for raw data on the socket, 2004 for pickled data
use_pickle 1 ; default value is 0, 1 for pickled data
tick_limit 300 ; Default value 300
graphite_data_source shinken ; default is that the variable is unset
}
```

16.16.4 SQL Based Modules - PNP4Nagios Graphing

PNP4Nagios is a graphing tool that has a web interface for RRDTool based databases. Shinken can export performance data to an npcd database which will feed the RRD files (fixed sized round robin databases). You can *learn how to install PNP4Nagios* if you haven't done it already.

npcdmod: export perfdatas to a PNP interface

Export all perfdata for PNP.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: npcdmod

- path: path to the npcd config file

Here is an example:

```
define module{
 module_name  NPCD
 module_type  npcdmod
 config_file  /usr/local/pnp4nagios/etc/npcd.cfg
}
```

16.16.5 SQL Based Modules - Frontend Centreon2

Centreon2 use a NDO database, so you will need the ndodb_mysql module for it.

ndodb_mysql: Export to a NDO Mysql database

Export status into a NDO/Mysql database. It needs the python module MySQLdb (apt-get install python-mysqldb or easy_install MySQL-python).

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: ndodb_mysql
- database: database name (ex ndo)
- user: database user
- password: database user passworddt
- host: database host
- character_set: utf8 is a good one

Here is an example:

```
define module{
```

```
     module_name      ToNdodb_Mysql
     module_type      ndodb_mysql
     database         ndo       ; database name
     user             root      ; user of the database
     password         root      ; must be changed
     host             localhost ; host to connect to
     character_set    utf8      ;optionnal, UTF8 is the default
}
```

16.16.6 File based Legacy modules - perfdata export

service_perfdata: export service perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: service_perfdata
- path: path to the service perfdata file you want

Here is an example:

```
define module{
```

```
     module_name      Service-Perfdata
     module_type      service_perfdata
     path             /usr/local/shinken/src/var/service-perfdata
}
```

host_perfdata: export host perfdatas to a flat file

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: host_perfdata
- path: path to the host perfdata file you want

Here is an example:

```
define module{
```

```
     module_name      Host-Perfdata
     module_type      host_perfdata
     path             /usr/local/shinken/src/var/host-perfdata
}
```

16.16.7 Legacy File based Modules - Old CGI or VShell

The Old CGI and VShell use the flat file export. If you can, avoid it! It has awful performance, all modern UIs no longer use this.

status_dat: export status into a flat file

Export all status into a flat file in the old Nagios format. It's for small/medium environment because it's very slow to parse. It can be used by the Nagios CGI. It also export the objects.cache file for this interface.

It takes the parameters:

- module_name: name of the module called by the brokers
- module_type: status_dat
- status_file: path of the status.dat file /usr/local/shinken/var/status.dat
- object_cache_file: path of the objects.cache file
- status_update_interval: interval in seconds to generate the status.dat file

Here is an example:

```
define module{
```

```
     module_name              Status-Dat
     module_type              status_dat
     status_file              /usr/local/shinken/var/status.data
     object_cache_file        /usr/local/shinken/var/objects.cache
```

```
    status_update_interval   15 ; update status.dat every 15s
}
```

16.16.8 Deprecated or unsupported modules

SQL Based MerlinDB

This interface is no longer supported and is going to be completely removed from Shinken.

SQL based ndodb_oracle - Icinga web frontend using NDO Oracle

This interface is no longer supported and is going to be completely removed from Shinken.

CouchDB: Export to a Couchdb database

This interface is no longer supported and is going to be completely removed from Shinken.

16.17 TSCA (Thrift Service Check Acceptor)

TSCA is a receiver/arbiter module to receive hosts and services check results. It provides the same functions as NSCA with additional features. High performance binary transmission, multi line output and auto generated clients for a variety of languages.

16.17.1 What is thrift ?

Let's cut and paste the wikipedia definition from'en.wikipedia.org/wiki/Apache_Thrift'_ :

"Thrift is an interface definition language that is used to define and create services for numerous languages.[1] It is used as a remote procedure call (RPC) framework [..]. It combines a software stack with a code generation engine to build services that work efficiently to a varying degree and seamlessly between ActionScript, C#, C++ (on POSIX-compliant systems), Cappuccino, Cocoa, Erlang, Haskell, Java, OCaml, Perl, PHP, Python, Ruby, and Smalltalk."

16.17.2 Installation

In any case, you have to install thrift with support for python (on the server side) and languages you want to use for clients.

Get it from thrift.apache.org or in your favorite distribution repository.

Server side

Go to the tsca module directory and generate the python stub for the server side installation

```
cd shinken/modules/tsca
$ thrift --gen py tsca.thrift
```

Declare the module is shinken-specific.org

```
define module{
      module_name     TSCA
      module_type     tsca_server
      host            0.0.0.0
      port            9090
}
```

Samples clients

There are three(3) client samples written in python, ruby and java in contrib/clients/TSCA of the Shinken distribution. You can use and extend them to send data directly in your code directly to Shinken's TSCA receiver/arbiter module.

The samples clients read a CSV file from stdin and send them the the TSCA running on localhost:9090. The CSV has the following format :

```
hostname,service description,output,rc
```

Any use of TCP in an application should take care not to block the program and to have very short timeouts. For applications that need non blocking functions, prefer a UDP based transport or a local buffer file with a separate application(send_nsca, thrift, etc) to forward your data to Shinken.

Ruby

Generate the stub

```
$ thrift --gen rb ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | ruby RubyClient.rb
```

Python

Generate the stub

```
$ thrift --gen py ../../../shinken/modules/tsca/tsca.thrift
```

Run the client

```
$ cat file_state | python PythonClient.py
```

java

Generate the stub

```
$ thrift --gen java ../../../shinken/modules/tsca/tsca.thrift
```

Compile the client

```
$ ant
```

Run the client

```
$ cat file_state | java JavaClientThrift
```

16.18 VMWare Arbiter module

16.18.1 How it works

The arbiter has a module that allows the host->VM automatic host dependency creation. In fact, the module is far more generic than just this. It's a generic "host link dependencies" module.

It reads a mapping file (simple json format) with son->father links. It call an external script at a regular interval : the one I wrote is a VMWare mapping one. It used the check_esx3.pl (that all ESX admin should have) plugin and generate the mapping file. When this file is changed, the module reloads it, creates new links and deletes old ones. So when you VMotion a virtual machine, in the next pass, there will be creation of a new link and deletion of the old one in the scheduler that manages it :)

16.18.2 How to define it

The definition is very easy :

```
define module{
     module_name       VMWare_auto_linking
     module_type       hot_dependencies
     mapping_file      /tmp/vmware_mapping_file.json
     mapping_command   /usr/local/shinken/libexec/link_vmware_host_vm.py -x '/usr/local/shinken/libex
     mapping_command_interval 60   ; optionnal
     mapping_command_timeout   300 ; optionnal
}
```

Then you add it in you arbiter object :

```
define arbiter {
    [...]
    modules   VMWare_auto_linking
}
```

So here, it will call the script /usr/local/shinken/libexec/link_vmware_host_vm.py with a listing of the vCenter vcenter.mydomain.com and generate the mapping file /tmp/vmware_mapping_file.json.

You can call : /usr/local/shinken/libexec/link_vmware_host_vm.py –help

for the full help on this script.

It looks at the file modification time each second. This command can also be launch by an external cron of course.

Note: So if you want to write a script that does this for XEN servers, all you need is to generate a json file with all the mapping. I didn't wrote the doc about this, but you can look in the test file for it, the format is very very easy :)

Here is an example of the JSON mapping file syntax:

%%:ref:*["host", "physical_host"], ["host", "virtual_machine"* <["host", "physical_host"], ["host", "virtual_machine">, "host", "physical_host"], ["host", "other_virtual_machine" <"host", "physical_host"], ["host", "other_virtual_machine">]%%

So just a list of couple 'host' string (from now only host are managed) and the host name.

Beware about it : you need the real host_name, including case.

16.19 Web Service Module

The Ws_arbiter module is a Receiver or Arbiter module to receive passive check results from remote hosts via HTTP(s). This module is considered a work in progress but can be used for testing and improvement.

16.19.1 Configuring the Web Service Module

If data is POSTed to this page, it will validate the data to determine if it is a service or host check result and submit the data to the Shinken external command queue.

The web service listens by default to TCP port 7760. The web service supports anonymous or authenticated access. Web service parameters are set in the Shinken-specific.cfg file, in the Ws_arbiter module. To enable the web service, simply add the module to your Arbiter or Receiver daemon configuration in Shinken-specific.cfg

16.19.2 Using the Web Service Module

The web service listens for POSTs to: /push_check_result

Use curl or embed the HTTP calls in your software to submit check results.

curl -u user:password -d "time_stamp=$(date +%s)&host_name=host-checked&service_description=service-checked&return_code=0&output=Everything OK" http://shinken-srv:7760/push_check_result

Example with more readability:

```
curl
```

-u user:password -d "time_stamp=$(date +%s)

```
&host_name=host-checked
&service_description=service-checked
&return_code=0
&output=Everything OK
http://shinken-srv:7760/push_check_result
```

16.20 SNMP module

This part must be splitted and sent in all module repositories

16.20.1 Shinken integrated SNMP Module

SnmpBooster Download Install and Configure

- *What is the SnmpBooster module*
- Install and configure the SNMP acquisition module. [You are here]
- *Reference - SnmpBooster troubleshooting*
- *Reference - SnmpBooster design specification*
- *Reference - SnmpBooster configuration dictionnary*

Downloads

The SnmpBooster module and genDevConfig are currently in public beta prior to integration within Shinken. You can consult t

- github.com/xkilian/genDevConfig
- github.com/titilambert/shinken/tree/snmp_booster/shinken/modules (snmp_booster branch) * Simply right-click and download snmp_poller.py then copy it to shinken/modules/

Requirements

The SnmpBooster module requires:

- Python 2.6+
- Shinken 1.2+
- PySNMP 4.2.1+ (Python module and its dependencies)
- ConfigObj (Python module)
- python-memcached
- memcachedb or memcached package for your operating system (ex. For Ubuntu: apt-get install memcachedb) *Note*

The genDevConfig profile generator depends on:

- Perl 5.004+
- 4 perl modules available from CPAN and network repositories. genDevConfig/INSTALL has the installation details.

STRONGLY RECOMMENDED: Use the same version of Python and Pyro on all hosts running Shinken processes.

Installation

SnmpBooster:

- Install the dependencies
- Copy the file snmp_poller.py from the git repository to your shinken/modules directory.
- Configuration steps are listed in the present web page.

genDevConfig:

- Download and extract the archive to your server.
- See genDevConfig/INSTALL on how to install and configure it.

Configuration

How to define the SnmpBooster module in the Shinken daemons

You need to modify shinken-specific.cfg, which is located in $shinken/etc/shinken-specific.cfg

Arbiter daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

Scheduler daemon configuration

Simply declare the module:

<code>modules SnmpBooster

Poller daemon configuration Simply declare the module:

```
modules SnmpBooster</code>
```

SnmpBooster Module declaration

Included in Shinken v1.2.1 shinken-specific.cfg.

<code>define module {

```
module_name          SnmpBooster
module_type          snmp_poller
datasource           /usr/local/shinken/etc/packs/network/SnmpBooster/   ; SET THE DIRECTORY FOR YOUR
memcached_host       x.x.x.x  ; SET THE IP ADDRESS OF YOUR memcached SERVER
memcached_port       11211  ; default port for a memcached process
```

}

If you do not know the IP adress on which your memcache is listening, check under /etc/memcached.conf. Or do a:

```
netstat -a | grep memcached</code>
If you are running a test on the local machine you can leave memcached on 127.0.0.1 (localhost), but
```

How to define a Host and Service

Step 1

Create a template for your SNMP enabled devices.

Sample template:

<code>cd shinken/etc/packs/network/ mkdir SnmpBooster

vi shinken/etc/packs/network/SnmpBooster/templates.cfg

```
define command {
```

```
command_name    check_snmp_booster
command_line    check_snmp_booster -H $HOSTNAME$ -C $SNMPCOMMUNITYREAD$ -V 2c -t $ARG1$ -i $_SERVICE
module_type     snmp_poller
```

}

```
define service {
```

```
name                    default-snmp-template
check_command           check_snmp_booster!$_SERVICEDSTEMPLATE$!$_SERVICEINST$!$_SERVICETRIGGERGROUP
_inst                   None
_triggergroup           None
max_check_attempts      3
check_interval          1
retry_interval          1
register                0
```

}

```
define host{
```

```
name                    SnmpBooster-host
alias                   SnmpBooster-host template
check_command           check_host_alive
max_check_attempts      3
check_interval          1
retry_interval          1
use                     generic-host
register                0
```

}

Step 2 Define some hosts and services. You would typically use genDevConfig or another configuration generator to create these for you.

Mandatory service arguments related to SNMP polling:

```
_dstemplate                Cisco-Generic-Router  ; Name of a DSTEMPLATE defined in the SnmpBooster co
```

```
snmpcommunityread    which is set in your resource.cfg file
```

Optional service arguments related to SNMP polling with default values:

```
_inst                   None   ; Could be numeric: 0, None or an instance mapping function like: map
```

```
_triggergroup           None   ; Name of the triggergroup defined in the SnmpBooster config.ini file
```

Sample Shinken host and service configuration:

```
# Generated by genDevConfig 3.0.0
# Args: --showunused -c publicstring 192.168.2.63
# Date: Thu Aug 30 17:47:59 2012

#######################################################################
# Description: Cisco IOS Software, C2960 Software (C2960-LANBASEK9-M), Version 12.2(50)SE4, RELEASE S
#     Contact:
# System Name: SITE1-ASW-Lab04
```

```
#    Location:
#######################################################################

define host {
```

```
 host_name            192.168.2.63
 display_name         192.168.2.63
 _sys_location
 address              192.168.2.63
 hostgroups
 notes
 parents
 use                  default-snmp-host-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis
 display_name         C2960 class chassis
 _dstemplate          Cisco-Generic-Router
 _inst                0
 use                  default-snmp-template
 register             1
}

define service {
```

```
 host_name            192.168.2.63
 service_description  chassis.device-traffic
 display_name         Switch fabric statistics - Packets per Second
 _dstemplate          Device-Traffic
 use                  default-snmp-template
 register             1
}

define service {
```

```
host_name            192.168.2.63
service_description  if.FastEthernet0_1
display_name         FastEthernet0_1 Description: Link to Router-1 100.0 MBits/s ethernetCsmacd
_dstemplate          standard-interface
_inst                map(interface-name,FastEthernet0_1)
use                  default-snmp-template
register             1
```

}

Here is an example configuration of the config.ini file

```
[DATASOURCE]
```

```
  OidmyOidDefinition = .1.3.6.1.45.0
  [myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
      ds_type = DERIVE
      ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = ds_calc
```

```
      ds_oid = $OidmyOidDefinition
[DSTEMPLATE]
```

```
  [myCiscoRouter]
      ds = myOidDefinition
[TRIGGER]
```

```
  [trigger1]
      warning = RPN expression
      critical = RPN expression
  [trigger2]
      warning = RPN expression
      critical = RPN expression
[TRIGGERGROUP]
```

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

16.20.2 Shinken integrated SNMP Module

Overview

What is it

The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

- Jump back to *SnmpBooster documentation index*

Design specification summary

- STATUS - DESIGN SPEC PERFORMANCE
- [Done] Functions as an integrated Shinken Poller module
- [Done] Necessary integration code commited to Shinken official release (Integrated starting at v1.2)
- [Done] Ability to collect thousands of SNMP metrics per second
- [Done] Be compatible with distributed data acquisition
- [Done] Collect data for a host/check_interval tuple via SNMP in a single pass
- [Done] Use all builtin Shinken scheduler logic for retries, forced checks, timeouts, dependencies, parents
- [Done] Store collected data for the duration of the check_interval in a memcached/membase
- [Done] On a restart, after the first collection, be able to pick up where the last check left and calculate derived values
- [Done] Forced check are not allowed within 30 seconds of last SNMP query to the same host/check_interval, all other requests get data from the cache.
- [Done] Only a single request to the host/check_interval via SNMP is allowed at a time, all other requests get data from the cache.

- STATUS - DESIGN SPEC USABILITY
- [Done] Usage documentation
- [xxxx] SkonfUI and Discovery usage documentation
- [xxxx] Provide sample configuration packs in Shinken
- [Done] Provide sample config.ini with examples of all types of data * SNMP OIDS, DATASOURCES, DSTEMPLATES, TRIGGERS and TRIGGERGROUPS
- [Done] Directly compatible for use with genDevConfig Shinken SNMP configuration generator
- [Done] Provide meaningful feedback for users on errors
- [Done] Capture all tracebacks and convert them to actual error or warning messages
- STATUS - DESIGN SPEC FEATURES
- [Done] Return state and performance metrics
- [Done] Performance metrics can be returned in a Weathermap compatible format
- [Done] Configuration file format is ConfigObj INI
- [Done] Load all valid INI configuration files from a directory and merge them
- [Done] Load a single INI configuration file
- [xxxx] Load a list of INI configuration files
- [Done] Configuration file describes all generic acquisition parameters (OID, DATASOURCE, DSTEMPLATE, MAP, TRIGGER, TRIGGERGROUP)
- [Done] Supports Triggers which are calculation rules to determine states
- [Done] Triggers support an RPN (Reverse Polish Notation) calculation engine which includes mathematical and logical operators
- [Done] Each TRIGGER is associated with a severity level, WARNING or CRITICAL
- [Done] Multiple TRIGGERS can be associated with a TRIGGERGROUP
- [Done] Use builtin Python Operators
- [Done] Support DERIVE, TEXT, GAUGE and COUNTER data types
- [xxxx] Support TIMETICKS data type
- [Done] Support applying RPN based calculations to received metric for scaling or conversion purposes
- [Done] Use a Python SNMP library which supports asynchronous acquisition PySNMP
- [Done] Datasources can use rule based runtime instance mapping
- [Done] Set Snmp version as a check runtime option
- [Done] Set Snmp DSTEMPLATE as a check runtime option
- [Done] Set Snmp TRIGGERGROUP as a check runtime option
- [Done] Set Snmp COMMUNITY as a check runtime option
- [Done] Use Snmp version 2c GetBulk
- [xxxx] Support Snmp version 2c GetNext if GetBulk is not supported
- [xxxx] Support Snmp version 1 GetNext

- [xxxx] Set Snmp Timeout as a check runtime option, instead of a hardcoded value at 5 seconds
- STATUS - DESIGN SPEC MAINTAINABILITY
- [xxxx] Functions documented in the source code
- [Done] Critical functions documented in the source code
- [Done] Locking sections identified in the code
- [xxxx] Unit tests with at least 80% coverage
- [xxxx] Unit tests integrated with Shinken test suite
- [Done] Code hosted on github
- [xxxx] configuration validity and integrity checking of all INI files
- [xxxx] Pep8 compliant
- [xxxx] Pylint pass

genDevConfig Plugins - Compatibility status with SnmpBooster

- STATUS - genDevConfig maintained Plugins
- [Done] Avaya ES switches
- [Done] Avaya ERS routing switches
- [Done] Cisco 29x0 switches
- [Done] MIB-II Interfaces
- [Done] Cisco PIX/ASA
- [Done] JUNOS devices
- [Done] Cisco IOS routers
- [Done] NetSNMP unix hosts ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Sensatronics devices ** Validation required**
- [Done] Foundry devices ** Validation required**
- [Done] Packeteer devices ** Validation required**
- [Done] Cisco CSS ** Validation required**
- [InProgress] New Cisco Access points
- STATUS - Other maintained Plugins

Tip: [xxxx] Denotes a specification that is planned but not implemented

[InProgress] Denotes a specification that is under development

[Done] Denotes a specification that is implemented

16.20.3 SnmpBooster reference dictionary

- Jump back to *SnmpBooster documentation index*

SnmpBooster.ini dictionary

There are five dictionaries:

- [DATASOURCE]
- [DSTEMPLATE]
- [MAP]
- [TRIGGER]
- [TRIGGERGROUP]

DATASOURCE DICTIONARY

OidVariableName refers to an actual OID that can be queries using SNMP against the device on the network.

[VariableName] refers to a Datasource and all the information required to gather and prepare the data using SNMP ds_type refers to how the data should be prepared ds_calc refers to any scaling manipulations to make the data more understandable. This is an RPN expression, where the first variable is omitted, as it is always the $OidVariable ds_oid refers to the actual $OidVariable name. An instance identifier can be appended to the name to signify that an instance is provided by the Shinken service definition. This information is passed when the check is called. ...

DSTEMPLATE DICTIONNARY

[DsTemplateName] refers to the name of the DSTEMPLATE that will be referred to in the Shinken service definitions. ds refers to the list of DATASOURCES to be collected. If an instance is expected for the list of DATASOURCES, it MUST be the same instance for all Oids. If a different instance is required, use a second DSTEMPLATE.

TRIGGER DICTIONNARY

TRIGGERGROUP DICTIONNARY

MAP DICTIONNARY

SnmpBooster.ini example configuration

This example definition will be used to explain each section.

[DATASOURCE]

```
OidmyOidDefinition = .1.3.6.1.45.0
[myOidDefinition] ; Use the same name as the myOidDeiniftion, but omit the leading "Oid"
    ds_type = DERIVE
    ds_calc = 8,*  ; RPN expression : Oid, 8, *  Which means Oid * 8 = Total
    ds_oid = $OidmyOidDefinition
```

[DSTEMPLATE]

```
[myCiscoRouter]
    ds = myOidDefinition
```

[TRIGGER]

```
[trigger1]
    warning = RPN expression
    critical = RPN expression
[trigger2]
    warning = RPN expression
    critical = RPN expression
```

[TRIGGERGROUP]

```
[CiscoRouterTriggers]
    triggers = trigger1, trigger2
```

SnmpBooster.ini configuring SNMP Datasources

The first location is generic data related to SNMP parameters.

- DATASOURCE information * SNMP OID * Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK) * Data format preparation (Scaling the data for example bits to bytes) * Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function

SnmpBooster.ini configuring SNMP DSTEMPLATES

- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service

SnmpBooster.ini setting triggers/thresholds

- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

16.20.4 Shinken integrated SNMP Module

Overview

- Jump back to *SnmpBooster documentation index*

Tip: The module is now in public beta as of 2012-10-26! Have Fun!

Nothing new to add, more to come in 2013. 2013-01-19.

What is it

The SnmpBooster module allows Shinken Pollers to directly manage SNMP data acquisition. This is an all Python cross-platform SNMP module. It is tightly integrated with the Shinken Poller, Scheduler and Arbiter daemons to provide the best possible user experience.

Why use it

The SnmpBooster module is designed to be very efficient and scalable. It has a very flexible configuration method to make it easy to use with Shinken Monitoring Packs and Shinken discovery runners like genDevConfig.

This acquisition module was professionally designed and developed.

It is meant to be used by the very capable discovery engine genDevConfig (**v3.0.5 and newer**) originally developed for the Cricket SNMP monitoring tool and converted for use with Shinken.

It is one of the very few serious SNMP v2c implementation making use of SnmpGetBulk type requests.

How does it work

Shinken Integration

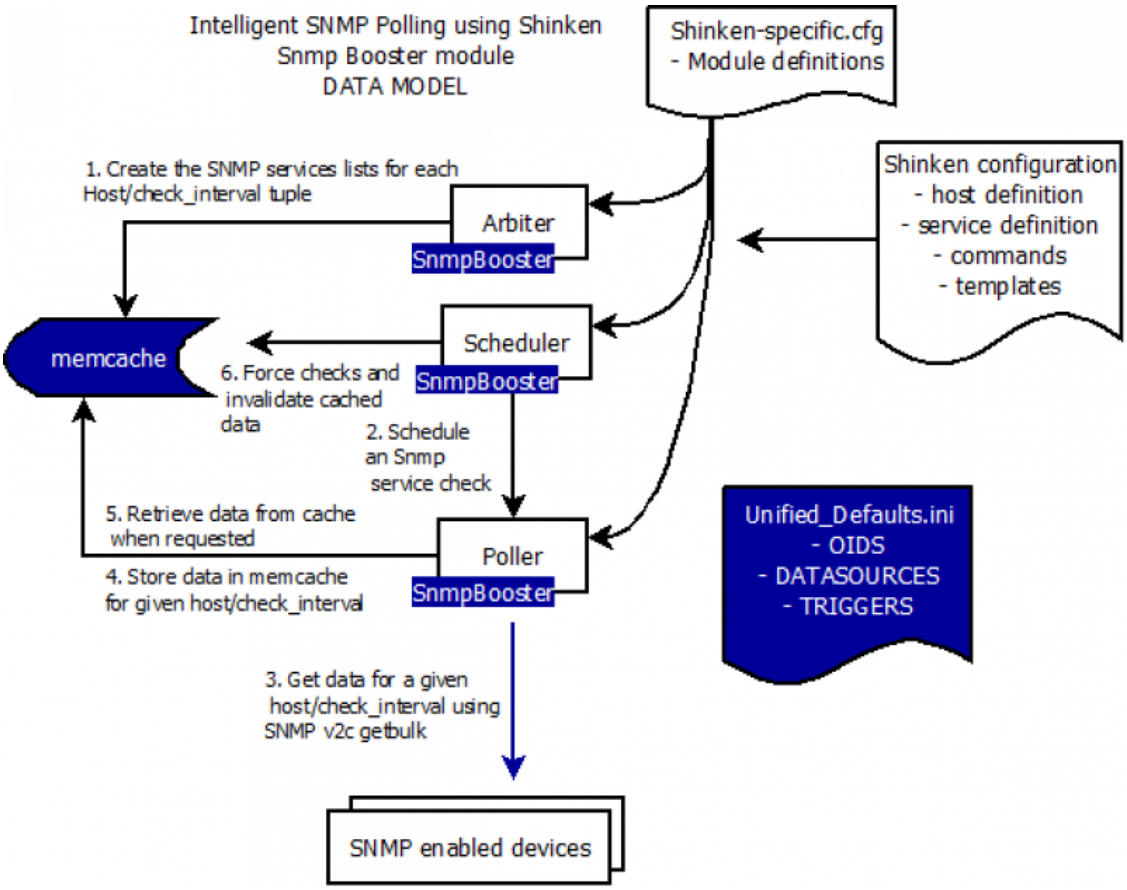

1. The SnmpBooster Arbiter module reads the Shinken SnmpBooster configuration file(s). It reads the check commands and based on the values in the check commands that use the snmp_poller module it will creates a shared configuration cache using memcached/memcachedb. The memcache key is always the combination of the hostname. This permits to tie together Shinken Hosts and Services with the SNMP specific configuration. (MongoDB and Redis are planned to be supported in the future) The Scheduler daemon schedules Host and Service checks as it normally does.

3. The SnmpBooster module will check that a memcache host key exists for the check to be launched. Then the SnmpBooster poller module will query all services related to a host for a given acquisition frequency in one pass. (all checks done every 2 minutes will be executed in bulk, same for those every 5 minutes)

5. Check results are cached for the acquisition check_interval. The next time a check request for data from the same host is received from the Scheduler, the data will be fetched from memcache. Resulting is extremely low check latency.

2. 6. The SNMP checks still benefit from retries, timeouts, forced checks, dependencies and the powerful Scheduler daemon logic.

6. The Scheduler daemon can still force an immediate re-check by flushing the cache. This is required for the dependency model.

Performance

SnmpBooster uses SNMP v2c getbulk for high efficiency, unless forced to use SNMP v2c get-next or SNMPv1 get-next. GetBulk uses a single request PDU to ask for multiple OIDs or even entire tables, instead of sending one request PDU per OID.

For example: *A typical 24 port network switch with two uplinks might use 375 OIDS (8 OIDs per interface, plus general OIDs for chassis health, memory, cpu, fans, etc.). **SnmpBooster will only require around 4 request PDUs instead of 375 request PDUs**.* Each PDU is a request packet which takes time to create, send get processed and return. More timeouts to manage, more connections, more impact on the remote device and more latency means much fewer checks per second.*

The SnmpBooster module supports automatic instance mapping for OIDs. (Ex. Based on the interface name it will figure out that the SNMP index(or instance) is 136. This is automatically handled by genDevConfig and SnmpBooster, no user input required. :-)

The generic SNMP configuration information is stored in the Shinken SnmpBooster INI files. There is a Defaults_unified.ini and a series of other Defaults files, one per discovery plugin for genDevConfig.

Important: genDevConfig plugins have all been converted to use the new dynamic instance mapping methods. You are now free to use most if not all Defaults*.ini files included with genDevConfig. 2012-10-28

Limitations

You should have memcached instances running on each poller responsible for snmp polling. You should have your pollers with SnmpBooster in the same datacenter, as they need to be on the same machine with good connectivity to the active memcached instance. SnmpBooster is not compatible with distributed pollers in multiple datacenters, sorry, the current design of SnmpBooster uses a single centralized memcached instance for storing the timeseries data. For distributed datacenters to be supported, each poller+scheduler+memcached must be realm restrained, which is not the case today.

Design specification

SnmpBooster design specification and current development status.

Data model

The information required to define the data is split in two locations.

The first location is the host and service Shinken configuration (You need to generate or write this)

- Device specific information
  - IP addresses, host_names, device types, instance keys

  - A DSTEMPLATE must be referred to in the Service definition
  - A static SNMP instance could be referred to in the Service definition
  - An SNMP instance MAP function could be referred to in the Service definition
  - A TRIGGERGROUP could be refered to in the Service definition

The second location is SNMP Defaults.* templates. (Here you can create new devices or add new data sources)

- DATASOURCE information
  - SNMP OID
  - Type of data and how can it be interpreted (GAUGE, COUNTER, COUNTER64, DERIVE, DERIVE64, TEXT, TIMETICK)
  - Data format preparation (Scaling the data for example bits to bytes)
  - Is there an instance to append to the
- Instance MAP function * Mapping the instance dynamically using a function * Data or rules related to the mapping function
- DSTEMPLATEs to associate DATASOURCE to actual device classes * List of DATASOURCES associated with a, for example, Cisco 1900 router. Which in turn can be applied to a Shinken service
- TRIGGER and TRIGGERGROUPS to apply thresholding rules * Define triggers and associate them with a TRIGGERGROUP name that can be applied to a Shinken Service

A final location containes rules to build your Shinken configuration.

- genDevConfig plugins create Shinken configurations

Installation and configuration

SnmpBooster installation

Reference Dictionnary

SnmpBooster reference dictionary

Troubleshooting

SnmpBooster troubleshooting

Graph templates

These are .graph files defined in your Shinken configuration directory. Refer to the Shinken graphite templates(Not yet created) or PNP4Nagios how-to documentation. The graph templates are independent from SnmpBooster and provide templates for any collected data from Shinken.

16.20.5 SnmpBooster Troubleshooting

- Jump back to *SnmpBooster documentation index*

Check your config

- Have you defined the poller module name?
- Have you defined the correct path to the directory containing your Defaults*.ini files?
- Have you addded the Snmpbooster module to your arbiter, poller, scheduler?
- Have you added copied the genDevConfig templates.cfg in shinken/packs/network/SnmpBooster/
- Have you installed PySNMP, memcached and other dependencies?

Software version consistency

Shinken and SnmpBooster now require **the same Python and Pyro version on all hosts running a Shinken daemon**.

If you cannot use the packaged version of Python and its modules (Pyro, memcached, etc.). Use virtualenv to declare a python version to use and install all required modules in that virtualenv.

Software version requirements

Have you verified that the *requirements* are met. Python, PySNMP, Shinken, Pyro, memcached, etc.

Validate your check command arguments

Use the check_plugin command and comment out the module to learn what were the exact arguments sent by the poller. This will permit you to validate all the arguments, like snmp community string, inheritance, template application, etc.

Validate connectivity

Take a packet trace using a tool like Wireshark to validate that the remote host is responding.

- Has the host responded
- **Is SnmpBooster repeating the request more often than the polling interval.**
  - If you are seeing repeated requests your device may have a compatibility issues.
  - Save an snmpwalk of the device, get a packet trace using Wireshark, set the poller to debug and save the poller.log file (shinken/var/poller.log). Send all three to the SnmpBooster developers.
  - Note: It is normal to see one or more bulkGet requests if you are getting large amounts of data. Ex. a 24 port switch will take 2-3 request packets.

Performance

Make sure you have a low latency connection to your memcache from the Poller. Your memcache server can be replicated to all your poller hosts that should also run memcache instances. Check that memcached is running: netstat -a | grep memcached

Faulty Template

A bad snmp_template file was distributed in the genDevConfig sample-config directory, there were two glaring errors.

This was fixed on 2012-10-16. Make sure you update your template, or use the data from the wiki.

Note that the template should be called: SnmpBooster-template.cfg to make it easier to troubleshoot in the logs. So when you search for SnmpBooster in your logs it will show up as well.

Log files

All warnings and errors generated by the SnmpBooster module start with "[SnmpBooster] error text" and are logged using the standard Shinken logger.

The Arbiter daemon can output initial configuration, loading of host keys and intervals in memcached type error messages. The Scheduler daemon can output scheduling and alert related messages. The Poller daemon can output messages related to instance mapping, acquisition timeouts, invalid community strings, cache failures and more. These are available in the Web interface, as they are placed in the check results for the service.

You can simply do a "grep SnmpBooster *" in your shinken/var directory to see the latest messages related to the SnmpBooster module. You can also sort messages by timestamp to make it easy to find where and when errors occurred.

cd shinken/var grep SnmpBooster *

memcache persistence

If you restart your memcache server or memcache crashes, your poller will no longer be able to validate that a host exists in memcache prior to writing.

You should use memcachedb to achieve persistence in case of memcache failures.

Common errors returned by SnmpBooster in the log file

Errors should be fairly explicit and mean what they say, though there can be exceptions. Lets try to clear some of them.

__Arbiter log **errors**__

Missing ds_oid : This means that a variable in your OID definitions is missing, or your DATASOURCE is not named correctly or your ds_oid variable is missing. There is a typo in your ds_oid variable (ex. ds-oid, or ds_oid = $OidNameIncorrectFFFRA.%(instance)s).

Datasource not defined : Your DSTEMPLATE uses a DATASOURCE that doesn't exist check the [DataSourceName] you are referring to. Does it contain the expected OID variable, $OidName.

Missing ds_type : The DATASOURCE always needs to have a ds_type definition, GAUGE, COUNTER, DERIVE, TEXT, TIMETICK, DERIVE64, COUNTER64.

__Poller log **errors**__

Problems with calculations, repeated polling, hosts not responding, etc.

__Memcached **errors**__

memcachedb and memcached do not use the same default port. Configure the correct memcachedb port to match what is declared in your SnmpBooster module under shinken-specific.cfg.

On Ubuntu 12.04 the default installation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when Shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

16.20.6 Memcached Notes

On Ubuntu 12.04 the default instalation is on port 21201 instead of 11211. This causes the error "[SnmpBooster] Memcache server (127.0.0.1:11211) is not reachable" when shinken starts.

To change it, you must edit the file /etc/memcachedb.conf

16.21 WebUI module

This part must be splitted and sent in all module repositories

16.21.1 WebUI integration with Shinken

The WebUI is an external broker module. So it's an external process launched by the broker that "eats" broks (broker messages) from the brokers. So it can get *all* broks if you place this broker in your top level realm.

The WebUI does not use a database to get data. Instead, it use a "regenerator" object that will eat each broks, and will "recreate" all Hosts/Services/Contacts objects as if the code was in the scheduler daemon. So basically, WebUI and pages always reflect the CURRENT state of hosts/services objects.

The WebUI http layer is managed by bottle.py. It's a small but powerful framework.

How does the WebUI function globally ?

There a 2 Threads running:

- the main thread is the bottle.py (http) listener. It will use the best backend it finds. We recommend using python-paste. If it does not find one, it will use the default swsgi. Paste is multi-thread, swsgi is mon-thread, so please use paste if you want to avoid locking problems.
- the 2nd thread is the regenerator one. It only gets broks and gives them to the regenerator object.

There is a lock system between the threads. You can have N Http requests in progress without problem, but during this time the renererator one will wait. During execution of the regenerator, no requests are processed. This is not considered a problem, as the regenerator is very quick, and there is only a 1s lock for a 7k service conf loading, so it's not a huge problem.

What is a WebUI User?

A WebUI user is a contact. A true contact object. You must match a contact name for your user.

To create a new page?

All pages are in the shinken/webui/plugins directory. Each "part" has it's own directory. A part (like the eltdetail, element detail) can have more than one "page" (like /host or /service for eltdetail).

From here we will supose you want to create a "/contact" page.

You can copy the dummy example in the plugins directory into a contact directory.

You need to change the contact/dummy.py file into a contact/contact.py file, because the webui will search for a dir/dir.py file to "load".

Templates

Templates are HTML with some Python code in it in the plugins/*/views direcotry for specific templates, and the "global layout" (menu and all common part) in the webui/views directory.

Your template must call the template layout (layout.tpl in the webui/views directory) wit ha "rebase" call.

If you put the rebase call at the beginning of your template, all there is below will be place "inside" the layout that will call the common javascripts and css, make the menu adn then call your html/python code.

The rebase call can take arguments. You MUST give it:

- user: the user you give it in the return dict of your .py code

Then all is optional:

- title: title of the page
- top_right_banner_state: int, global state with clouds to put in your top right page. Look at the problem.tpl code for an example.
- js and css list: list of YOUR specific javascript or css to call AFTER the common ones.
- refresh: boolean, set the refresh to 60s or do not refresh at all
- menu_part: will put the little "^" under this part of the menu.

Example of a call, taken from the problems view:

```
rebase layout title='All problems', top_right_banner_state=top_right_banner_state, js=['problems/js/a
roblems/js/autocompleter.Request.js', 'problems/js/autocompleterObserver.js'], css=['problems/css/acc
ms/css/autocompleter.css', 'problems/css/perfometer.css'], refresh=True, menu_part='/'+page, user=use
```

Call Python objects in your templates

Your templates are mainly HTML code. But of course you can call python code in it. The dict you returns in the plugin.py call will be available in this page.

For example, the my_host variable will be available in the page with the host object you find (or None if your give it a bad name of course). Then you can "call" python with for example here to show the host_name: `<span>My host name is *my_host.host_name* </span>` is a way in the templates to "call a variable or function" that will return a string, here the host_name.

If the function or property you call must return HTML code that SHOULD be output as it, without "protection", you must put a ! before your call.

Let use an example where the output value of an host is in fact an html output. If you use the code `my_host.output`, bottle.py will "protect" the html generated by escaping the string, and change <> characters with escaped value. You can change this by calling `my_host.output`. Of course, it's a security problem if the output is with a javascript code...

You can also "loop" or put "if" in your template. You just need to put your code with a % before. Like

```
%for i in xrange(1, 10):
  My valus is  {{i}}
%end
```

You must put the %end here, the indentation will NOT be enougth, sorry.

For an if it's also easy:

```
%if host.active_checks_enabled:
  The host got active checks enabled
%else:
  The host DO NOT have active check enabled
%end
```

Which javascript lib are you using?

It's mootools 1.3 with all options :)

What to do?

- Finish the /mobile part

- List of current alerts
- Host info
- Host status
- Host interfaces
- Hostgroup status
- dashboard hierachique with states, and BP
- Nagvis
- add bp in nagvis auto-map
- event sources
- correlation
- pre-filtering/enrichment
- user profile
- host/srv conf

16.21.2 WebUI search syntax

work in progress

not that we couldn't tell. ;) eh eh

beware the working men and flying tools!

16.21.3 WebUI SSL Certification

Lets start: the goal to to have the server.pem and client.pem file in etc/certs.

Go to the etc/certs directory.

```
openssl req -new -nodes -out server.req -keyout server.key -config ./shinken_openssl_cnf
openssl ca -out server.pem -config ./shinken_openssl_cnf -infiles server.req
cp server.pem server.pem.bkp
cp server.key server.pem
cat server.pem.bkp >> server.pem
cp server.pem client.pem
```

You can try to get the validity with for the arbiter with:

```
openssl s_client -connect localhost:7770</code>
```

CHAPTER 17

Reference

17.1 shinken

17.1.1 shinken Package

shinken Package

acknowledge Module

class shinken.acknowledge.**Acknowledge**(*ref*, *sticky*, *notify*, *persistent*, *author*, *comment*, *end_time=0*)

Allows you to acknowledge the current problem for the specified service. By acknowledging the current problem, future notifications (for the same servicestate) are disabled.

id = 1

properties = {'comment': None, 'author': None, 'sticky': None, 'end_time': None, 'id': None, 'notify': None}

action Module

class shinken.action.**Action**

Bases: shinken.action.__Action

execute__(*force_shell=False*)

kill__()

shinken.action.**no_block_read**(*output*)

arbiterlink Module

autoslots Module

The AutoSlots Class is a MetaClass: it manages how other classes are created (Classes, not instances of theses classes). Here it's role is to create the __slots__ list of the class with all properties of Class.properties and Class.running_properties so we do not have to add manually all properties to the __slots__ list when we add a new entry

class shinken.autoslots.**AutoSlots**

Bases: type

basemodule Module

This python module contains the class BaseModule that shinken modules will subclass

class shinken.basemodule.**BaseModule**(*mod_conf*)

Bases: object

This is the base class for the shinken modules. Modules can be used by the different shinken daemons/services for different tasks. Example of task that a shinken module can do:

- load additional configuration objects.
- recurrently save hosts/services status/perfdata informations in different format.
- ...

clear_queues(*manager*)

Release the resources associated to the queues of this instance

create_queues(*manager=None*)

The manager is None on android, but a true Manager() elsewhere Create the shared queues that will be used by shinken daemon process and this module process. But clear queues if they were already set before recreating new one.

do_loop_turn()

For external modules only: implement in this method the body of you main loop

do_stop()

Called just before the module will exit Put in this method all you need to cleanly release all open resources used by your module

get_name()

has(*prop*)

The classic has: do we have a prop or not?

init()

Handle this module "post" init ; just before it'll be started. Like just open necessaries file(s), database(s), or whatever the module will need.

main()

module "main" method. Only used by external modules.

manage_brok(*brok*)

Request the module to manage the given brok. There a lot of different possible broks to manage.

manage_signal(*sig, frame*)

set_exit_handler(*sigs=None*)

set_loaded_into(*daemon_name*)

set_proctitle(*name*)

set_signal_handler(*sigs=None*)

start()

stop_process()

Request the module process to stop and release it

want_brok(*b*)

work()

module "main" method. Only used by external modules.

class shinken.basemodule.**ModulePhases**
TODO: Add some comment about this class for the doc

CONFIGURATION = 1

LATE_CONFIGURATION = 2

RETENTION = 8

RUNNING = 4

bin Module

borg Module

class shinken.borg.**Borg**
Bases: object

Here is the new-style Borg (not much more complex then the "old-style")

brok Module

class shinken.brok.**Brok**(*type*, *data*)
A Brok is a piece of information exported by Shinken to the Broker. Broker can do whatever he wants with it.

id = 0

my_type = 'brok'

prepare()

brokerlink Module

check Module

class shinken.check.**Check**(*status*, *command*, *ref*, *t_to_go*, *dep_check=None*, *id=None*, *timeout=10*, *poller_tag='None'*, *reactionner_tag='None'*, *env={}*, *module_type='fork'*, *from_trigger=False*, *dependency_check=False*)
Bases: *shinken.action.Action*

ODO: Add some comment about this class for the doc

check_time

check_type

copy_shell()
return a copy of the check but just what is important for execution So we remove the ref and all

dep_check

depend_on

env

execution_time

exit_status

from_trigger

get_id()

get_return_from(*c*)

internal

is_a

is_dependent()

is_launchable(*t*)

long_output

module_type

my_type = 'check'

output

perf_data

poller_tag

properties = {'check_type': <shinken.property.IntegerProp object at 0x7fa5c5c8a710>, 'exit_status': <shinken.proper

reactionner_tag

ref

s_time

set_type_active()

set_type_passive()

state

status

t_to_go

type

u_time

worker

commandcall Module

class shinken.commandcall.**CommandCall**(*commands*, *call*, *poller_tag='None'*, *reactionner_tag='None'*)

Bases: *shinken.commandcall.DummyCommandCall*

This class is use when a service, contact or host define a command with args.

args

call

command

get_command_and_args()

We want to get the command and the args with ! splitting. but don't forget to protect against the ! to do not split them

get_name()

id = 0

is_valid()

late_linkify_with_command(*commands*)

late_relink_done

module_type

my_type = 'CommandCall'

poller_tag

properties = {'poller_tag': <shinken.property.StringProp object at 0x7fa5c5ba0b10>, 'reactionner_tag': <shinken.pro

reactionner_tag

timeout

valid

class shinken.commandcall.**DummyCommandCall**

Bases: object

Ok, slots are fun: you cannot set the __autoslots__ on the same class you use, fun isn't it? So we define* a dummy useless class to get such :)

comment Module

class shinken.comment.**Comment**(*ref*, *persistent*, *author*, *comment*, *comment_type*, *entry_type*, *source*, *expires*, *expire_time*)

id = 1

properties = {'comment': None, 'source': None, 'entry_time': None, 'comment_type': None, 'author': None, 'expire_

complexexpression Module

class shinken.complexexpression.**ComplexExpressionFactory**(*ctx='hostgroups'*, *grps=None*, *all_elements=None*)

Bases: object

eval_cor_pattern(*pattern*)

find_object(*pattern*)

class shinken.complexexpression.**ComplexExpressionNode**

Bases: object

is_valid()

resolve_elements()

contactdowntime Module

class shinken.contactdowntime.**ContactDowntime**(*ref*, *start_time*, *end_time*, *author*, *comment*)

cancel()

check_activation()

enter()

exit()

id = 1

in_scheduled_downtime()

properties = {'comment': None, 'author': None, 'start_time': None, 'is_in_effect': None, 'end_time': None, 'can_be_c

daemon Module

daterange Module

class shinken.daterange.**CalendarDaterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

class shinken.daterange.**Daterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

get_min_from_t(*t*)

get_min_sec_from_morning()

get_min_sec_out_from_morning()

classmethod get_month_by_id(*id*)

classmethod get_month_id(*month*)

get_next_future_timerange_invalid(*t*)

get_next_future_timerange_valid(*t*)

get_next_invalid_day(*t*)

get_next_invalid_time_from_t(*t*)

get_next_valid_day(*t*)

get_next_valid_time_from_t(*t*)

get_start_and_end_time(*ref=None*)

classmethod get_weekday_by_id(*id*)

classmethod get_weekday_id(*weekday*)

is_correct()

is_time_day_invalid(*t*)

is_time_day_valid(*t*)

is_time_valid(*t*)

months = {'march': 3, 'august': 8, 'september': 9, 'december': 12, 'june': 6, 'july': 7, 'november': 11, 'february': 2, 'oct

weekdays = {'monday': 0, 'tuesday': 1, 'friday': 4, 'wednesday': 2, 'thursday': 3, 'sunday': 6, 'saturday': 5}

class shinken.daterange.**MonthDateDaterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

class shinken.daterange.**MonthDayDaterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

class shinken.daterange.**MonthWeekDayDaterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

is_correct()

class shinken.daterange.**StandardDaterange**(*day*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

is_correct()

class shinken.daterange.**Timerange**(*entry*)

get_first_sec_out_from_morning()

get_sec_from_morning()

is_time_valid(*t*)

class shinken.daterange.**WeekDayDaterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

shinken.daterange.**find_day_by_offset**(*year*, *month*, *offset*)

shinken.daterange.**find_day_by_weekday_offset**(*year*, *month*, *weekday*, *offset*)

db Module

class shinken.db.**DB**(*table_prefix=''*)

Bases: object

DB is a generic class for SQL Database

create_insert_query(*table*, *data*)

Create a INSERT query in table with all data of data (a dict)

create_update_query(*table*, *data*, *where_data*)

Create a update query of table with data, and use where data for the WHERE clause

fetchone()

Just get an entry

stringify(*val*)
Get a unicode from a value

db_mysql Module

class shinken.db_mysql.**DBMysql**(*host*, *user*, *password*, *database*, *character_set*, *table_prefix=''*, *port=3306*)
Bases: shinken.db.DB

DBMysql is a MySQL access database class

connect_database()
Create the database connection TODO: finish (begin :)) error catch and conf parameters... Import to catch exception

execute_query(*query*, *do_debug=False*)
Just run the query TODO: finish catch

db_oracle Module

class shinken.db_oracle.**DBOracle**(*user*, *password*, *database*, *table_prefix=''*)
Bases: shinken.db.DB

Manage connection and query execution against Oracle databases.

connect_database()
Create the database connection TODO: finish (begin :)) error catch and conf parameters...

execute_query(*query*)
Execute a query against an Oracle database.

db_sqlite Module

class shinken.db_sqlite.**DBSqlite**(*db_path*, *table_prefix=''*)
Bases: shinken.db.DB

DBSqlite is a sqlite access database class

connect_database()
Create the database connection

execute_query(*query*)
Just run the query

dependencynode Module

class shinken.dependencynode.**DependencyNode**
Bases: object

get_reverse_state(*state*)

get_state()

is_valid()

list_all_elements()

switch_zeros_of_values()

class shinken.dependencynode.**DependencyNodeFactory**

Bases: object

eval_cor_pattern(*pattern*, *hosts*, *services*)

find_object(*pattern*, *hosts*, *services*)

dispatcher Module

This is the class of the dispatcher. Its role is to dispatch configurations to other elements like schedulers, reactionner, pollers, receivers and brokers. It is responsible for high availability part. If an element dies and the element type has a spare, it sends the config of the dead one to the spare

class shinken.dispatcher.**Dispatcher**(*conf*, *arbiter*)

check_alive()

check_bad_dispatch()

check_dispatch()

dispatch()

get_scheduler_ordered_list(*r*)

downtime Module

class shinken.downtime.**Downtime**(*ref*, *start_time*, *end_time*, *fixed*, *trigger_id*, *duration*, *author*, *comment*)

add_automatic_comment()

cancel()

del_automatic_comment()

enter()

exit()

fill_data_brok_from(*data*, *brok_type*)

get_initial_status_brok()

id = 1

in_scheduled_downtime()

properties = {'comment': <shinken.property.StringProp object at 0x7fa5c44a7550>, 'entry_time': <shinken.property.

trigger_me(*other_downtime*)

easter Module

shinken.easter.**dark**()

.-.

|_:_|

/(_Y_)

```
            ( /M/ )
      '. _.'-/'-'-'._
            ': _/.-'[[[[]'-._
                ': /_'  [|::''|][`.\]
                        ': // ./ |oUU| .' :
                          ': _:'..' \_|___|_/  [:|]
                              ':. .' |_[____]_| :.':
                                [::\ |  [| |][;][\] '-' /'.| |.'  .;.' | |_ '-' : | | .: : | | | | '. : | / :. .; |
                                / | | :__/ : \


                              | | : | | ||


                            / : : |: / |__| /|

                    snd |  [: :_/_| /'._\ '-|_]
                          /___.-/_|-'   '-'
```

shinken.easter.**episode_iv**()

shinken.easter.**get_coffee**()

```
                                                (
                                                  ) (  ____...(------)-....___
                                            .-"" ) ( ""-.
                                        .-'''|-._ ) _.-|
                                  / .-.| " "-.........-" " |
                            / / | | | | | |
                                  | |
                                      | |
                                                ''| | _/ / /
                                            (__/ /
                            _..-""'' /''""-..._
                      .-' / '-.
            : '-.__ __.-' : : ) ""-...-"" ( :
                  '._  "-...____...-"  _.'
      jgs ""-..__ __..-""/
            '._ ""-.....______.....-""  _.'
                  ""-..,,_____ _____,,..-""  "" ''__ "" ''
```

shinken.easter.**myip**()

shinken.easter.**naheulbeuk**()

shinken.easter.**perdu**()

eventhandler Module

class shinken.eventhandler.**EventHandler**(*command*, *id=None*, *ref=None*, *timeout=10*, *env={}*, *module_type='fork'*, *reactionner_tag='None'*)

Bases: *shinken.action.Action*

check_time

command

copy_shell()

env

execution_time

exit_status

get_id()

get_outputs(*out*, *max_plugins_output_length*)

get_return_from(*e*)

is_a

is_launchable(*t*)

long_output

module_type

my_type = 'eventhandler'

output

perf_data

properties = {'status': <shinken.property.StringProp object at 0x7fa5c42ece50>, 'module_type': <shinken.property.St

reactionner_tag

s_time

sched_id

status

t_to_go

timeout

type

u_time

worker

external_command Module

graph Module

class shinken.graph.**Graph**

Graph is a class to make graph things like DFS checks or accessibility Why use an atomic bomb when a little hammer is enough?

add_edge(*from_node*, *to_node*)

add_node(*node*)

add_nodes(*nodes*)

dfs_get_all_childs(*root*)

dfs_loop_search(*root*)

get_accessibility_packs()

loop_check()

http_client Module

http_daemon Module

load Module

class shinken.load.**Load**(*m=1*, *initial_value=0*)

This class is for having a easy Load calculation without having to send value at regular interval (but it's more efficient if you do this :)) and without having a list or other stuff. It's just an object, an update and a get You can define m: the average for m minutes. The val is the initial value. It's better if it's 0 but you can choose.

get_load()

update_load(*new_val*, *forced_interval=None*)

log Module

class shinken.log.**Log**

Shinken logger class, wrapping access to Python logging standard library.

CRITICAL = 50

DEBUG = 10

ERROR = 40

INFO = 20

NOTSET = 0

WARNING = 30

critical(*msg*, **args*, ***kwargs*)

debug(*msg*, **args*, ***kwargs*)

error(*msg*, **args*, ***kwargs*)

get_level()

static get_level_id(*lvlName*)
Convert a level name (string) to its integer value and vice-versa. Input a level and it will return a name. Raise KeyError when name or level not found

info(*msg*, **args*, ***kwargs*)

load_obj(*object*, *name_=None*)
We load the object where we will put log broks with the 'add' method

log(*message*, *format=None*, *print_it=True*)
Old log method, kept for NAGIOS compatibility What strings should not use the new format ??

quit()
Close the local log file at program exit

register_local_log(*path*, *level=None*)
The shinken logging wrapper can write to a local file if needed and return the file descriptor so we can avoid to close it.

set_human_format(*on=True*)
Set the output as human format.

If the optional parameter *on* is False, the timestamps format will be reset to the default format.

set_level(*level*)

warning(*msg*, **args*, ***kwargs*)

shinken.log.**cprint**(*s*, *color*)

shinken.log.**is_tty**()

macroresolver Module

class shinken.macroresolver.**MacroResolver**
Bases: *shinken.borg.Borg*

Please Add a Docstring to describe the class here

get_env_macros(*data*)

init(*conf*)

macros = {'TOTALSERVICESCRITICALUNHANDLED': '_get_total_services_critical_unhandled', 'PROCESSSTART

my_type = 'macroresolver'

resolve_command(*com*, *data*)

resolve_simple_macros_in_string(*c_line*, *data*, *args=None*)

memoized Module

class shinken.memoized.**memoized**(*func*)
Bases: object

Decorator that caches a function's return value each time it is called. If called later with the same arguments, the cached value is returned, and not re-evaluated.

message Module

class shinken.message.**Message**(*id*, *type*, *data=None*, *source=None*)

This is a simple message class for communications between actionners and workers

get_data()

get_from()

get_type()

my_type = 'message'

str()

modulesctx Module

modulesmanager Module

class shinken.modulesmanager.**ModulesManager**(*modules_type*, *modules_path*, *modules*)

Bases: object

This class is use to manage modules and call callback

check_alive_instances()

clear_instances(*insts=None*)

get_external_from_queues()

get_external_instances(*phase=None*)

get_external_to_queues()

get_instances()

get_internal_instances(*phase=None*)

load()

load_and_init()

load_manager(*manager*)

remove_instance(*inst*)

set_max_queue_size(*max_queue_size*)

set_modules(*modules*)

set_to_restart(*inst*)

start_external_instances(*late_start=False*)

stop_all()

try_instance_init(*inst*, *late_start=False*)

try_to_restart_deads()

notification Module

class shinken.notification.**Notification**(*type='PROBLEM', status='scheduled', command='UNSET', command_call=None, ref=None, contact=None, t_to_go=0, contact_name='', host_name='', service_description='', reason_type=1, state=0, ack_author='', ack_data='', escalated=False, contacts_notified=0, start_time=0, end_time=0, notification_type=0, id=None, notif_nb=1, timeout=10, env={}, module_type='fork', reactionner_tag='None'*)

Bases: *shinken.action.Action*

Please Add a Docstring to describe the class here

ack_author

ack_data

already_start_escalations

check_time

command

command_call

contact

contact_name

contacts_notified

copy_shell()

creation_time

end_time

env

escalated

execution_time

exit_status

fill_data_brok_from(*data*, *brok_type*)

get_id()

get_initial_status_brok()

get_return_from(*n*)

host_name

is_a

is_administrative()

is_launchable(*t*)

macros = {'NOTIFICATIONAUTHORNAME': 'author_name', 'HOSTNOTIFICATIONNUMBER': 'notif_nb', 'NOTI

module_type

my_type = 'notification'

notif_nb

notification_type

output

properties = {'reason_type': <shinken.property.StringProp object at 0x7fa5c3e16dd0>, 'status': <shinken.property.St

reactionner_tag

reason_type

s_time

sched_id

service_description

start_time

state

status

t_to_go

timeout

type

u_time

worker

pollerlink Module

property Module

class shinken.property.**UnusedProp**(*text=None*)

Bases: shinken.property.Property

A unused Property. These are typically used by Nagios but no longer useful/used by Shinken.

This is just to warn the user that the option he uses is no more used in Shinken.

class shinken.property.**BoolProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

A Boolean Property.

Boolean values are currently case insensitively defined as 0, false, no, off for False, and 1, true, yes, on for True).

pythonize(*val*)

class shinken.property.**IntegerProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

Please Add a Docstring to describe the class here

pythonize(*val*)

class shinken.property.**FloatProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

Please Add a Docstring to describe the class here

pythonize(*val*)

class shinken.property.**CharProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

Please Add a Docstring to describe the class here

pythonize(*val*)

class shinken.property.**StringProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

Please Add a Docstring to describe the class here

pythonize(*val*)

class shinken.property.**ListProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

Please Add a Docstring to describe the class here

pythonize(*val*)

reactionnerlink Module

receiverlink Module

satellite Module

satellitelink Module

scheduler Module

schedulerlink Module

singleton Module

class shinken.singleton.**Singleton**(*name*, *bases*, *dict*)

Bases: type

The classic Singleton class. So all instance of this class will be the same instance in fact.

sorteddict Module

class shinken.sorteddict.**SortedDict**(**args*, ***kwargs*)

Bases: dict

Dictionary with sorted keys.

The interface is similar to the ordinary dict's one, but: * methods: __repr__(), __str__(), __iter__(), iterkeys(), itervalues(),

iteritems(), keys(), values(), items() and popitem() – return results taking into consideration sorted keys order;

•new methods: largest_key(), largest_item(), smallest_key(), smallest_item() added.

clear() → None. Remove all items from D.

copy() → a shallow copy of D (still as a SortedDict).

classmethod fromkeys(*S*[, *v*]) → New dict with keys from S and values equal to v.

v defaults to None.

items() → a list of (key, value) pairs sorted by keys

(add reverse=True for reverse ordering).

iteritems() → an iterator over (key, value) pairs sorted by keys

(add reverse=True for reverse ordering).

iterkeys(*reverse=False*)

D.__iter__() <==> iter(D) <==> D.iterkeys() -> an iterator over sorted keys (add reverse=True for reverse ordering).

itervalues() → an iterator over values sorted by keys

(add reverse=True for reverse ordering).

keys() → a sorted list of keys

(add reverse=True for reverse ordering).

largest_item() → a (key, value) pair with the largest key;

raise KeyError if D is empty.

largest_key() → the largest key; raise KeyError if D is empty.

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v). Remove and return a (key, value) pair with
the largest key; raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

smallest_item() → a (key, value) pair with the smallest key;
raise KeyError if D is empty.

smallest_key() → the smallest key; raise KeyError if D is empty.

update([*E*], **F) → None. Update D from dict/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → a list of values sorted by keys
(add reverse=True for reverse ordering).

shinken.sorteddict.**dictdoc**(*method*)
A decorator making reuse of the ordinary dict's docstrings more concise.

trigger_functions Module

util Module

shinken.util.**alive_then_spare_then_deads**(*x*, *y*)

shinken.util.**apply_change_recursive_pattern_change**(*s*, *rule*)

shinken.util.**expand_with_macros**(*ref*, *value*)

shinken.util.**expand_xy_pattern**(*pattern*)

shinken.util.**expect_file_dirs**(*root*, *path*)

shinken.util.**format_t_into_dhms_format**(*t*)

shinken.util.**from_bool_to_int**(*b*)

shinken.util.**from_bool_to_string**(*b*)

shinken.util.**from_float_to_int**(*val*)

shinken.util.**from_list_to_split**(*val*)

shinken.util.**get_customs_keys**(*d*)

shinken.util.**get_customs_values**(*d*)

shinken.util.**get_day**(*t*)

shinken.util.**get_end_of_day**(*year*, *month_id*, *day*)

shinken.util.**get_key_value_sequence**(*entry*, *default_value=None*)

shinken.util.**get_obj_full_name**(*obj*)

shinken.util.**get_obj_name**(*obj*)

shinken.util.**get_obj_name_two_args_and_void**(*obj*, *value*)

shinken.util.**get_sec_from_morning**(*t*)

shinken.util.**get_start_of_day**(*year*, *month_id*, *day*)

shinken.util.**get_wday**(*t*)

shinken.util.**got_generation_rule_pattern_change**(*xy_couples*)

shinken.util.**nighty_five_percent**(*t*)

shinken.util.**print_date**(*t*)

shinken.util.**safe_print**(**args*)

shinken.util.**scheduler_no_spare_first**(*x*, *y*)

shinken.util.**sort_by_ids**(*x*, *y*)

shinken.util.**strip_and_uniq**(*tab*)

shinken.util.**to_best_int_float**(*val*)

shinken.util.**to_bool**(*val*)

shinken.util.**to_char**(*val*)

shinken.util.**to_float**(*val*)

shinken.util.**to_hostnames_list**(*ref*, *tab*)

shinken.util.**to_int**(*val*)

shinken.util.**to_list_of_names**(*ref*, *tab*)

shinken.util.**to_list_string_of_names**(*ref*, *tab*)

shinken.util.**to_name_if_possible**(*ref*, *value*)

shinken.util.**to_split**(*val*)

shinken.util.**to_svc_hst_distinct_lists**(*ref*, *tab*)

worker Module

class shinken.worker.**Worker**(*id*, *s*, *returns_queue*, *processes_by_worker*, *mortal=True*, *timeout=300*, *max_plugins_output_length=8192*, *target=None*, *loaded_into='unknown'*)

This class is used for poller and reactionner to work. The worker is a process launch by theses process and read Message in a Queue (self.s) (slave) They launch the Check and then send the result in the Queue self.m (master) they can die if they do not do anything (param timeout)

add_idletime(*time*)

check_for_system_time_change()

do_work(*s*, *returns_queue*, *c*)

get_new_checks()

id = **0**

is_alive()

is_killable()

is_mortal()

join(*timeout=None*)

launch_new_checks()

manage_finished_checks()

reset_idle()

send_message(*msg*)

set_proctitle()

set_zombie()

start()

terminate()

work(*s*, *returns_queue*, *c*)

Subpackages

clients Package

clients Package

LSB Module

class shinken.clients.LSB.**FileQueryGenerator**(*filename*)

Bases: *shinken.clients.LSB.SimpleQueryGenerator*

class shinken.clients.LSB.**QueryGenerator**

Bases: object

Generate a livestatus query

get()

class shinken.clients.LSB.**SimpleQueryGenerator**(*querys*, *name='sqg'*)

Bases: *shinken.clients.LSB.QueryGenerator*

get()

shinken.clients.LSB.**main**(*argv*)

shinken.clients.LSB.**mean**(*numberList*)

shinken.clients.LSB.**median**(*numberList*)

shinken.clients.LSB.**run**(*url*, *requests*, *concurrency*, *qg*)

shinken.clients.LSB.**usage**()

livestatus Module

class shinken.clients.livestatus.**LSAsynConnection**(*addr='127.0.0.1'*, *port=50000*, *path=None*, *timeout=10*)

Bases: asyncore.dispatcher

do_connect()

do_read(*size*)

exec_command(*command*)

get_query()

get_returns()

handle_close()

handle_connect()

handle_read()

handle_write()

is_finished()

launch_raw_query(*query*)

look_for_timeout()

readable()

stack_query(*q*)

wait_returns()

writable()

class shinken.clients.livestatus.**LSConnectionPool**(*con_addrs*)

Bases: object

launch_raw_query(*query*)

class shinken.clients.livestatus.**LSSyncConnection**(*addr='127.0.0.1', port=50000, path=None, timeout=10*)

connect()

exec_command(*command*)

launch_query(*query*)

read(*size*)

class shinken.clients.livestatus.**Query**(*q*)

Bases: object

get()

id = 0

put(*r*)

daemons Package

daemons Package

arbiterdaemon Module

brokerdaemon Module

pollerdaemon Module

reactionnerdaemon Module

receiverdaemon Module

schedulerdaemon Module

discovery Package

discovery Package

discoverymanager Module

misc Package

misc Package

datamanager Module

class shinken.misc.datamanager.**DataManager**

Bases: object

get_all_hosts_and_services()

get_all_impacts()

get_all_problems(*to_sort=True*, *get_acknowledged=False*)

get_brokers()

get_business_parents(*obj*, *levels=3*)

get_contact(*name*)

get_contacts()

get_host(*hname*)

get_host_tags_sorted()

get_hostgroup(*name*)

get_hostgroups()

get_hostgroups_sorted(*selected=''*)

get_hosts()

get_important_elements()

get_important_impacts()

get_len_overall_state()

get_nb_all_problems()

get_nb_elements()

get_nb_impacts()

get_nb_problems()

get_overall_it_state()

get_overall_state()

get_per_hosts_state()

get_per_service_state()

get_pollers()

get_problems_time_sorted()

get_program_start()

get_reactionners()

get_realm(*r*)

get_realms()

get_receivers()

get_schedulers()

get_service(*hname*, *sdesc*)

get_services()

guess_root_problems(*obj*)

load(*rg*)

filter Module Helper functions for some filtering, like for user based

shinken.misc.filter.**only_related_to**(*lst*, *user*)

md5crypt Module md5crypt.py - Provides interoperable MD5-based crypt() function

SYNOPSIS

import md5crypt.py

cryptedpassword = md5crypt.md5crypt(password, salt);

DESCRIPTION

unix_md5_crypt() provides a crypt()-compatible interface to the rather new MD5-based crypt() function found in modern operating systems. It's based on the implementation found on FreeBSD 2.2.[56]-RELEASE and contains the following license in it:

> "THE BEER-WARE LICENSE" (Revision 42): <phk@login.dknet.dk> wrote this file. As long as you retain this notice you can do whatever you want with this stuff. If we meet some day, and you think this stuff is worth it, you can buy me a beer in return. Poul-Henning Kamp

apache_md5_crypt() provides a function compatible with Apache's .htpasswd files. This was contributed by Bryan Hart <bryan@eai.com>.

shinken.misc.md5crypt.**apache_md5_crypt**(*pw*, *salt*)

shinken.misc.md5crypt.**md5crypt**(*pw*, *salt*, *magic=None*)

shinken.misc.md5crypt.**to64**(*v*, *n*)

shinken.misc.md5crypt.**unix_md5_crypt**(*pw*, *salt*, *magic=None*)

perfdata Module

class shinken.misc.perfdata.**Metric**(*s*)

class shinken.misc.perfdata.**PerfDatas**(*s*)

shinken.misc.perfdata.**guess_int_or_float**(*val*)

regenerator Module

sorter Module Helper functions for some sorting

shinken.misc.sorter.**hst_srv_sort**(*s1*, *s2*)

shinken.misc.sorter.**last_state_change_earlier**(*s1*, *s2*)

shinken.misc.sorter.**worse_first**(*s1*, *s2*)

termcolor Module ANSII Color formatting for output in terminal.

shinken.misc.termcolor.**colored**(*text*, *color=None*, *on_color=None*, *attrs=None*)
Colorize text.
Available text colors: red, green, yellow, blue, magenta, cyan, white.
Available text highlights: on_red, on_green, on_yellow, on_blue, on_magenta, on_cyan, on_white.
Available attributes: bold, dark, underline, blink, reverse, concealed.
Example: colored('Hello, World!', 'red', 'on_grey', ['blue', 'blink']) colored('Hello, World!', 'green')

shinken.misc.termcolor.**cprint**(*text*, *color=None*, *on_color=None*, *attrs=None*, ***kwargs*)
Print colorize text.

It accepts arguments of print function.

objects Package

objects Package

The objects package contains definition classes of the different objects that can be declared in configuration files.

businessimpactmodulation Module

class shinken.objects.businessimpactmodulation.**Businessimpactmodulation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

my_type = 'businessimpactmodulation'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

class shinken.objects.businessimpactmodulation.**Businessimpactmodulations**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *Businessimpactmodulation*

linkify(*timeperiods*)

linkify_cm_by_tp(*timeperiods*)

name_property = 'business_impact_modulation_name'

checkmodulation Module

class shinken.objects.checkmodulation.**CheckModulation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_check_command(*t_to_go*)

get_name()

id = 1

is_correct()

late_linkify_cw_by_commands(*commands*)

macros = {}

my_type = 'checkmodulation'

properties = {'check_period': <shinken.property.StringProp object at 0x7fa5c436d490>, 'use': <shinken.property.List

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.checkmodulation.**CheckModulations**(*items*)

Bases: *shinken.objects.item.Items*

inner_class

alias of *CheckModulation*

linkify(*timeperiods*, *commands*)

name_property = 'checkmodulation_name'

new_inner_member(*name=None*, *params={}*)

command Module

class shinken.objects.command.**Command**(*params={}*)

Bases: *shinken.objects.item.Item*

command_line

command_name

fill_data_brok_from(*data*, *brok_type*)

get_initial_status_brok()

get_name()

id = 0

imported_from

module_type

my_type = 'command'

name

poller_tag

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

pythonize()

reactionner_tag

timeout

use

class shinken.objects.command.**Commands**(*items*)

Bases: *shinken.objects.item.Items*

inner_class

alias of *Command*

name_property = 'command_name'

class shinken.objects.command.**DummyCommand**
Bases: object

config Module

contact Module

class shinken.objects.contact.**Contact**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

get_notification_commands(*type*)

id = 1

is_correct()

macros = {'CONTACTPAGER': 'pager', 'CONTACTGROUPNAMES': 'get_groupnames', 'CONTACTGROUPNAME'

my_type = 'contact'

old_properties = {'min_criticity': 'min_business_impact'}

properties = {'address1': <shinken.property.StringProp object at 0x7fa5c40b33d0>, 'address2': <shinken.property.St

raise_cancel_downtime_log_entry()

raise_enter_downtime_log_entry()

raise_exit_downtime_log_entry()

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'modified_att

want_host_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

want_service_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

class shinken.objects.contact.**Contacts**(*items*)
Bases: *shinken.objects.item.Items*

explode(*contactgroups*, *notificationways*)

inner_class
alias of *Contact*

late_linkify_c_by_commands(*commands*)

linkify(*timeperiods*, *commands*, *notificationways*)

linkify_with_notificationways(*notificationways*)

name_property = 'contact_name'

contactgroup Module

class shinken.objects.contactgroup.**Contactgroup**(*params={}*)
Bases: *shinken.objects.itemgroup.Itemgroup*

get_contactgroup_members()

get_contacts()

get_contacts_by_explosion(*contactgroups*)

get_name()

id = 1

macros = {'CONTACTGROUPALIAS': 'alias', 'CONTACTGROUPMEMBERS': 'get_members'}

my_type = 'contactgroup'

properties = {'contactgroup_name': <shinken.property.StringProp object at 0x7fa5c40c1510>, 'use': <shinken.proper

class shinken.objects.contactgroup.**Contactgroups**(*items*)

Bases: *shinken.objects.itemgroup.Itemgroups*

add_contactgroup(*cg*)

add_member(*cname*, *cgname*)

explode()

get_members_by_name(*cgname*)

inner_class

alias of *Contactgroup*

linkify(*contacts*)

linkify_cg_by_cont(*contacts*)

name_property = 'contactgroup_name'

discoveryrule Module

class shinken.objects.discoveryrule.**Discoveryrule**(*params={}*)

Bases: *shinken.objects.matchingitem.MatchingItem*

get_name()

id = 1

macros = {}

my_type = 'discoveryrule'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c4123950>}

class shinken.objects.discoveryrule.**Discoveryrules**(*items*)

Bases: *shinken.objects.item.Items*

inner_class

alias of *Discoveryrule*

name_property = 'discoveryrule_name'

discoveryrun Module

class shinken.objects.discoveryrun.**Discoveryrun**(*params={}*)

Bases: *shinken.objects.matchingitem.MatchingItem*

check_finished()

get_name()

get_output()

id = 1

is_finished()

is_first_level()

launch (*ctx=[], timeout=300*)

my_type = 'discoveryrun'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'discoveryrun_command': <shinken.prop

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.discoveryrun.**Discoveryruns** (*items*)

Bases: *shinken.objects.item.Items*

inner_class

alias of *Discoveryrun*

linkify (*commands*)

name_property = 'discoveryrun_name'

escalation Module

class shinken.objects.escalation.**Escalation** (*params={}*)

Bases: *shinken.objects.item.Item*

get_name ()

get_next_notif_time (*t_wished*, *status*, *creation_time*, *interval*)

id = 1

is_correct ()

is_eligible (*t*, *status*, *notif_number*, *in_notif_time*, *interval*)

my_type = 'escalation'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.escalation.**Escalations** (*items*)

Bases: *shinken.objects.item.Items*

add_escalation (*es*)

explode (*hosts*, *hostgroups*, *contactgroups*)

inner_class

alias of *Escalation*

linkify (*timeperiods*, *contacts*, *services*, *hosts*)

linkify_es_by_h (*hosts*)

linkify_es_by_s (*services*)

name_property = 'escalation_name'

host Module This is the main class for the Host. In fact it's mainly about the configuration part. for the running one, it's better to look at the schedulingitem class that manage all scheduling/consume check smart things :)

class shinken.objects.host.**Host** (*params={}*)

Bases: *shinken.objects.schedulingitem.SchedulingItem*

acknowledgement

acknowledgement_type

act_depend_of

act_depend_of_me

action_url

actions

active_checks_enabled

add_business_rule_act_dependency(*h*, *status*, *timeperiod*, *inherits_parent*)

add_host_act_dependency(*h*, *status*, *timeperiod*, *inherits_parent*)

add_host_chk_dependency(*h*, *status*, *timeperiod*, *inherits_parent*)

add_service_link(*service*)

address

alias

attempt

broks

business_impact

business_impact_modulations

business_rule

check_command

check_flapping_recovery_notification

check_freshness

check_interval

check_period

check_type

checkmodulations

checks_in_progress

child_dependencies

childs

chk_depend_of

chk_depend_of_me

comments

configuration_errors

configuration_warnings

contact_groups

contacts

current_event_id

current_notification_id

current_notification_number

current_problem_id

custom_views

customs

del_host_act_dependency(*other*)

display_name

downtimes

duration_sec

early_timeout

end_time

escalations

event_handler

event_handler_enabled

execution_time

failure_prediction_enabled

fill_parents_dependency()

fill_predictive_missing_parameters()

find_service_by_name(*service_description*)

first_notification_delay

flap_detection_enabled

flap_detection_options

flapping_changes

flapping_comment_id

freshness_threshold

get_ack_author_name()

get_ack_comment()

get_check_command()

get_data_for_checks()

get_data_for_event_handler()

get_data_for_notifications(*contact*, *n*)

get_dbg_name()

get_duration()

get_duration_sec()

get_full_name()

get_host_tags()

get_hostgroups()

get_name()

get_obsessive_compulsive_processor_command()

get_realm()

get_total_services()

get_total_services_critical()

get_total_services_ok()

get_total_services_unknown()

get_total_services_warning()

got_business_rule

got_default_realm

has_been_checked

hash

high_flap_threshold

host_name

hostgroups

icon_image

icon_image_alt

icon_set

id = 1

impacts

imported_from

in_checking

in_hard_unknown_reach_phase

in_scheduled_downtime

in_scheduled_downtime_during_last_check

initial_state

is_correct()

is_flapping

is_impact

is_linked_with_host(*other*)

is_problem

is_state(*status*)

last_chk

last_event_id

last_hard_state

last_hard_state_change

last_hard_state_id

last_notification

last_perf_data

last_problem_id

last_state

last_state_change

last_state_id

last_state_type

last_state_update

last_time_down

last_time_non_ok_or_up()

last_time_unreachable

last_time_up

latency

long_output

low_flap_threshold

macromodulations

macros = {'HOSTPERFDATA': 'perf_data', 'HOSTGROUPNAMES': 'get_groupnames', 'HOSTCHECKCOMMAND'

maintenance_period

manage_stalking(*c*)

max_check_attempts

modified_attributes

my_own_business_impact

my_type = 'host'

name

next_chk

notes

notes_url

notification_interval

notification_is_blocked_by_contact(*n*, *contact*)

notification_is_blocked_by_item(*type*, *t_wished=None*)

notification_options

notification_period

notifications_enabled

notifications_in_progress

notified_contacts

obsess_over_host

ok_up = 'UP'

old_properties = {'normal_check_interval': 'check_interval', 'hostgroup': 'hostgroups', 'criticity': 'business_impact

output

pack_id

parent_dependencies

parents

passive_checks_enabled

pending_flex_downtime

percent_state_change

perf_data

poller_tag

problem_has_been_acknowledged

process_perf_data

properties = {'statusmap_image': <shinken.property.StringProp object at 0x7fa5c3d58cd0>, 'icon_image_alt': <shink

raise_alert_log_entry()

raise_cancel_downtime_log_entry()

raise_enter_downtime_log_entry()

raise_event_handler_log_entry(*command*)

raise_exit_downtime_log_entry()

raise_flapping_start_log_entry(*change_ratio*, *threshold*)

raise_flapping_stop_log_entry(*change_ratio*, *threshold*)

raise_freshness_log_entry(*t_stale_by*, *t_threshold*)

raise_initial_state()

raise_no_next_check_log_entry()

raise_notification_log_entry(*n*)

reactionner_tag

realm

register_child(*child*)

resultmodulations

retain_nonstatus_information

retain_status_information

retry_interval

return_code

running_properties = {'state_id_before_impact': <shinken.property.StringProp object at 0x7fa5c3d4b550>, 'child_c

scheduled_downtime_depth

services

set_impact_state()

set_state_from_exit_status(*status*)

set_unreachable()

should_be_scheduled

source_problems

stalking_options

start_time

state

state_before_hard_unknown_reach_phase

state_before_impact

state_changed_since_impact

state_id

state_id_before_impact

state_type

state_type_id

statusmap_image

tags

time_to_orphanage

timeout

topology_change

trending_policies

trigger

trigger_name

triggers

unset_impact_state()

use

vrml_image

was_in_hard_unknown_reach_phase

class shinken.objects.host.**Hosts**(*items*)

Bases: *shinken.objects.item.Items*

apply_dependencies()

create_business_rules(*hosts*, *services*)

create_business_rules_dependencies()

explode(*hostgroups*, *contactgroups*, *triggers*)

fill_predictive_missing_parameters()

find_hosts_that_use_template(*tpl_name*)

inner_class

alias of *Host*

late_linkify_h_by_commands(*commands*)

linkify(*timeperiods=None*, *commands=None*, *contacts=None*, *realms=None*, *resultmodulations=None*, *businessimpactmodulations=None*, *escalations=None*, *hostgroups=None*, *triggers=None*, *checkmodulations=None*, *macromodulations=None*)

linkify_h_by_h()

linkify_h_by_hg(*hostgroups*)

linkify_h_by_realms(*realms*)

name_property = 'host_name'

no_loop_in_parents()

hostdependency Module

class shinken.objects.hostdependency.**Hostdependencies**(*items*)

Bases: *shinken.objects.item.Items*

apply_inheritance()

delete_hostsdep_by_id(*ids*)

explode(*hostgroups*)

linkify(*hosts*, *timeperiods*)

linkify_h_by_hd()

linkify_hd_by_h(*hosts*)

linkify_hd_by_tp(*timeperiods*)

class shinken.objects.hostdependency.**Hostdependency**(*params={}*)

Bases: *shinken.objects.item.Item*

get_name()

id = 0

my_type = 'hostdependency'

properties = {'inherits_parent': <shinken.property.BoolProp object at 0x7fa5c40c40d0>, 'use': <shinken.property.Lis

hostescalation Module

class shinken.objects.hostescalation.**Hostescalation**(*params={}*)

Bases: *shinken.objects.item.Item*

get_name()

id = 1

my_type = 'hostescalation'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

class shinken.objects.hostescalation.**Hostescalations**(*items*)

Bases: *shinken.objects.item.Items*

explode(*escalations*)

inner_class

alias of *Hostescalation*

name_property = ''

hostextinfo Module This is the main class for the Host ext info. In fact it's mainly about the configuration part. Parameters are merged in Hosts so it's no use in running part

class shinken.objects.hostextinfo.**HostExtInfo**(*params={}*)

Bases: *shinken.objects.item.Item*

get_dbg_name()

get_full_name()

get_name()

host_name

icon_image

icon_image_alt

id = 1

imported_from

is_correct()

macros = {'HOSTNOTES': 'notes', 'HOSTNAME': 'host_name', 'HOSTNOTESURL': 'notes_url'}

my_type = 'hostextinfo'

name

notes

notes_url

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

statusmap_image

use

vrml_image

class shinken.objects.hostextinfo.**HostsExtInfo**(*items*)

Bases: *shinken.objects.item.Items*

inner_class

alias of *HostExtInfo*

merge(*hosts*)

merge_extinfo(*host*, *extinfo*)

name_property = 'host_name'

hostgroup Module

class shinken.objects.hostgroup.**Hostgroup**(*params={}*)

Bases: *shinken.objects.itemgroup.Itemgroup*

get_hostgroup_members()

get_hosts()

get_hosts_by_explosion(*hostgroups*)

get_name()

id = 1

macros = {'HOSTGROUPMEMBERS': 'members', 'HOSTGROUPNOTES': 'notes', 'HOSTGROUPACTIONURL': 'a

my_type = 'hostgroup'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'hostgroup_name': <shinken.property.Str

class shinken.objects.hostgroup.**Hostgroups**(*items*)

Bases: *shinken.objects.itemgroup.Itemgroups*

add_member(*hname*, *hgname*)

explode()

get_members_by_name(*hgname*)

inner_class

alias of *Hostgroup*

linkify(*hosts=None*, *realms=None*)

linkify_hg_by_hst(*hosts*)

linkify_hg_by_realms(*realms*)

name_property = 'hostgroup_name'

item Module This class is a base class for nearly all configuration elements like service, hosts or contacts.

class shinken.objects.item.**Item**(*params={}*)

Bases: object

acknowledge_problem(*sticky*, *notify*, *persistent*, *author*, *comment*, *end_time=0*)

add_comment(*comment*)

add_downtime(*downtime*)

check_for_expire_acknowledge()

clean()

Clean useless things not requested once item has been fully initialized&configured. Like temporary attributes such as "imported_from", etc..

compute_hash()

copy()

Return a copy of the item, but give him a new id

del_comment(*comment_id*)

del_downtime(*downtime_id*)

explode_trigger_string_into_triggers(*triggers*)

fill_data_brok_from(*data*, *brok_type*)

fill_default()

Fill missing properties if they are missing

get_all_plus_and_delete()

get_check_result_brok()

get_customs_properties_by_inheritance(*items*)

get_initial_status_brok()

get_next_schedule_brok()

get_plus_and_delete(*prop*)

get_property_by_inheritance(*items*, *prop*)

get_property_value_for_brok(*prop*, *tab*)

get_raw_import_values()

get_templates()

get_update_status_brok()

has_plus(*prop*)

init_running_properties()

is_correct()

is_tpl()
Return if the elements is a template

linkify_one_command_with_commands(*commands*, *prop*)

linkify_with_triggers(*triggers*)

classmethod load_global_conf(*conf*)
Used to put global values in the sub Class like hosts or services

macros = {}

old_properties_names_to_new()

prepare_for_conf_sending()

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

pythonize()

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'hash': <shin

unacknowledge_problem()

unacknowledge_problem_if_not_sticky()

class shinken.objects.item.**Items**(*items*)
Bases: object

apply_inheritance()

apply_partial_inheritance(*prop*)

clean()
Request to remove the unnecessary attributes/others from our items

compute_hash()

create_reversed_list()

create_tpl_list()

evaluate_hostgroup_expression(*expr*, *hosts*, *hostgroups*, *look_in='hostgroups'*)

explode_contact_groups_into_contacts(*contactgroups*)

explode_host_groups_into_hosts(*hosts*, *hostgroups*)

explode_trigger_string_into_triggers(*triggers*)

fill_default()

find_by_name(*name*)

find_id_by_name(*name*)

find_tpl_by_name(*name*)

is_correct()

linkify_command_list_with_commands(*commands*, *prop*)

linkify_one_command_with_commands(*commands*, *prop*)

linkify_templates()

linkify_with_business_impact_modulations(*business_impact_modulations*)

linkify_with_checkmodulations(*checkmodulations*)

linkify_with_contacts(*contacts*)

linkify_with_escalations(*escalations*)

linkify_with_macromodulations(*macromodulations*)

linkify_with_resultmodulations(*resultmodulations*)

linkify_with_timeperiods(*timeperiods*, *prop*)

linkify_with_triggers(*triggers*)

old_properties_names_to_new()

prepare_for_sending()

pythonize()

remove_templates()

Remove useless templates (& properties) of our items ; otherwise we could get errors on config.is_correct()

remove_twins()

itemgroup Module

class shinken.objects.itemgroup.**Itemgroup**(*params={}*)

Bases: *shinken.objects.item.Item*

add_string_member(*member*)

copy_shell()

fill_default()

get_initial_status_brok()

has(*prop*)

id = 0

is_correct()

properties = {'members': <shinken.property.StringProp object at 0x7fa5c408a190>, 'unknown_members': <shinken.p

pythonize()

replace_members(*members*)

class shinken.objects.itemgroup.**Itemgroups**(*items*)

Bases: *shinken.objects.item.Items*

add(*ig*)

fill_default()

get_members_by_name(*gname*)

macromodulation Module

class shinken.objects.macromodulation.**MacroModulation**(*params={}*)

Bases: *shinken.objects.item.Item*

get_name()

id = 1

is_active()

is_correct()

macros = {}

my_type = 'macromodulation'

properties = {'macromodulation_name': <shinken.property.StringProp object at 0x7fa5c436d350>, 'use': <shinken.pr

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.macromodulation.**MacroModulations**(*items*)

Bases: *shinken.objects.item.Items*

inner_class

alias of *MacroModulation*

linkify(*timeperiods*)

name_property = 'macromodulation_name'

matchingitem Module This is a utility class for factorizing matching functions for discovery runners and rules.

class shinken.objects.matchingitem.**MatchingItem**(*params={}*)

Bases: *shinken.objects.item.Item*

is_matching(*key*, *value*, *look_in='matches'*)

is_matching_disco_datas(*datas*)

module Module

class shinken.objects.module.**Module**(*params={}*)

Bases: *shinken.objects.item.Item*

get_name()

id = 1

macros = {}

my_type = 'module'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'module_type': <shinken.property.String

class shinken.objects.module.**Modules**(*items*)

Bases: *shinken.objects.item.Items*

explode()

inner_class
alias of *Module*

linkify()

linkify_s_by_plug()

name_property = 'module_name'

notificationway Module

class shinken.objects.notificationway.**NotificationWay**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

get_notification_commands(*type*)

id = 1

is_correct()

late_linkify_nw_by_commands(*commands*)

macros = {}

my_type = 'notificationway'

old_properties = {'min_criticity': 'min_business_impact'}

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'notificationway_name': <shinken.proper

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

want_host_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

want_service_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

class shinken.objects.notificationway.**NotificationWays**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *NotificationWay*

linkify(*timeperiods*, *commands*)

name_property = 'notificationway_name'

new_inner_member(*name=None*, *params={}*)

pack Module

class shinken.objects.pack.**Pack**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

my_type = 'pack'

properties = {'pack_name': <shinken.property.StringProp object at 0x7fa5c3da7e50>, 'use': <shinken.property.ListP

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.pack.**Packs**(*items*)
Bases: *shinken.objects.item.Items*

create_pack(*buf*, *name*)

inner_class
alias of *Pack*

load_file(*path*)

name_property = 'pack_name'

realm Module

class shinken.objects.realm.**Realm**(*params={}*)

Bases: *shinken.objects.itemgroup.Itemgroup*

add_string_member(*member*)

count_brokers()

count_pollers()

count_reactionners()

count_receivers()

fill_broker_with_poller_reactionner_links(*broker*)

fill_potential_brokers()

fill_potential_pollers()

fill_potential_reactionners()

fill_potential_receivers()

get_all_subs_pollers()

get_all_subs_reactionners()

get_name()

get_nb_of_must_have_satellites(*type*)

get_potential_satellites_by_type(*type*)

get_realm_members()

get_realms()

get_realms_by_explosion(*realms*)

get_satellites_links_for_scheduler()

get_satellties_by_type(*type*)

id = 1

macros = {'REALMNAME': 'realm_name', 'REALMMEMBERS': 'members'}

my_type = 'realm'

prepare_for_satellites_conf()

properties = {'higher_realms': <shinken.property.StringProp object at 0x7fa5c408ad90>, 'use': <shinken.property.Lis

pythonize()

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.realm.**Realms**(*items*)

Bases: *shinken.objects.itemgroup.Itemgroups*

explode()

get_default()

get_members_by_name(*pname*)

inner_class
alias of *Realm*

linkify()

linkify_p_by_p()

name_property = 'realm_name'

prepare_for_satellites_conf()

resultmodulation Module

class shinken.objects.resultmodulation.**Resultmodulation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

module_return(*return_code*)

my_type = 'resultmodulation'

properties = {'exit_code_modulation': <shinken.property.StringProp object at 0x7fa5c3d59850>, 'use': <shinken.prop

pythonize()

class shinken.objects.resultmodulation.**Resultmodulations**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *Resultmodulation*

linkify(*timeperiods*)

linkify_rm_by_tp(*timeperiods*)

name_property = 'resultmodulation_name'

schedulingitem Module This class is a common one for service/host. Here you will find all scheduling related functions, like the schedule or the consume_check. It's a very important class!

class shinken.objects.schedulingitem.**SchedulingItem**(*params={}*)
Bases: *shinken.objects.item.Item*

add_attempt()

add_flapping_change(*b*)

check_and_set_unreachability()

check_for_flexible_downtime()

compensate_system_time_change(*difference*)

consume_result(*c*)

create_business_rules(*hosts*, *services*)

create_business_rules_dependencies()

create_notifications(*type*, *t_wished=None*)

current_event_id = **0**

current_problem_id = **0**

deregister_a_problem(*pb*)

disable_active_checks()

do_check_freshness()

do_i_raise_dependency(*status*, *inherit_parents*)

eval_triggers()

get_escalable_contacts(*n*)

get_event_handlers(*externalcmd=False*)

get_next_notification_time(*n*)

get_perfdata_command()

get_time_to_orphanage()

is_escalable(*n*)

is_max_attempts()

is_no_action_dependent()

is_no_check_dependent()

launch_check(*t*, *ref_check=None*, *force=False*)

manage_internal_check(*c*)

no_more_a_problem()

prepare_notification_for_sending(*n*)

raise_dependencies_check(*ref_check*)

rebuild_ref()

Rebuild the possible reference a schedulingitem can have

register_a_problem(*pb*)

register_son_in_parent_child_dependencies(*son*)

remove_in_progress_check(*c*)

remove_in_progress_notification(*n*)

remove_in_progress_notifications()

scatter_notification(*n*)

schedule(*force=False*, *force_time=None*)

set_myself_as_problem()

update_business_impact_value()

update_event_and_problem_id()

update_flapping()

update_hard_unknown_phase_state()

update_in_checking()

update_notification_command(*n*)

service Module This Class is the service one, s it manage all service specific thing. If you look at the scheduling part, look at the scheduling item class

class shinken.objects.service.**Service**(*params={}*)

Bases: *shinken.objects.schedulingitem.SchedulingItem*

acknowledgement

acknowledgement_type

act_depend_of

act_depend_of_me

action_url

actions

active_checks_enabled

add_business_rule_act_dependency(*srv*, *status*, *timeperiod*, *inherits_parent*)

add_service_act_dependency(*srv*, *status*, *timeperiod*, *inherits_parent*)

add_service_chk_dependency(*srv*, *status*, *timeperiod*, *inherits_parent*)

aggregation

attempt

broks

business_impact

business_impact_modulations

business_rule

check_command

check_flapping_recovery_notification

check_freshness

check_interval

check_period

check_type

checkmodulations

checks_in_progress

child_dependencies

chk_depend_of

chk_depend_of_me

comments

configuration_errors

configuration_warnings

contact_groups

contacts

current_event_id

current_notification_id

current_notification_number

current_problem_id

custom_views

customs

default_value

display_name

downtimes

duplicate(*host*)

duplicate_foreach

duration_sec

early_timeout

end_time

escalations

event_handler

event_handler_enabled

execution_time

failure_prediction_enabled

fill_daddy_dependency()

first_notification_delay

flap_detection_enabled

flap_detection_options

flapping_changes

flapping_comment_id

freshness_threshold

get_ack_author_name()

get_ack_comment()

get_check_command()

get_data_for_checks()

get_data_for_event_handler()

get_data_for_notifications(*contact*, *n*)

get_dbg_name()

get_duration()

get_duration_sec()

get_full_name()

get_groupnames()

get_host_tags()

get_hostgroups()

get_name()

get_obsessive_compulsive_processor_command()

get_realm()

got_business_rule

has_been_checked

hash

high_flap_threshold

host

host_name

hostgroup_name

icon_image

icon_image_alt

icon_set

id = 1

impacts

imported_from

in_checking

in_hard_unknown_reach_phase

in_scheduled_downtime

in_scheduled_downtime_during_last_check

initial_state

is_correct()

is_flapping

is_impact

is_problem

is_state(*status*)

is_volatile

last_chk

last_event_id

last_hard_state

last_hard_state_change

last_hard_state_id

last_notification

last_perf_data

last_problem_id

last_state

last_state_change

last_state_id

last_state_type

last_state_update

last_time_critical

last_time_non_ok_or_up()

last_time_ok

last_time_unknown

last_time_warning

latency

long_output

low_flap_threshold

macromodulations

macros = {'LASTSERVICESTATEID': 'last_state_id', 'SERVICEDURATION': 'get_duration', 'SERVICEGROUPNAM

maintenance_period

manage_stalking(*c*)

max_check_attempts

modified_attributes

my_own_business_impact

my_type = 'service'

name

next_chk

notes

notes_url

notification_interval

notification_is_blocked_by_contact(*n*, *contact*)

notification_is_blocked_by_item(*type*, *t_wished=None*)

notification_options

notification_period

notifications_enabled

notifications_in_progress

notified_contacts

obsess_over_service

ok_up = 'OK'

old_properties = {'normal_check_interval': 'check_interval', 'hostgroups': 'hostgroup_name', 'hostgroup': 'hostgro

output

parallelize_check

parent_dependencies

passive_checks_enabled

pending_flex_downtime

percent_state_change

perf_data

poller_tag

problem_has_been_acknowledged

process_perf_data

properties = {'active_checks_enabled': <shinken.property.BoolProp object at 0x7fa5c3d7a650>, 'icon_image_alt': <sh

raise_alert_log_entry()

raise_cancel_downtime_log_entry()

raise_enter_downtime_log_entry()

raise_event_handler_log_entry(*command*)

raise_exit_downtime_log_entry()

raise_flapping_start_log_entry(*change_ratio*, *threshold*)

raise_flapping_stop_log_entry(*change_ratio*, *threshold*)

raise_freshness_log_entry(*t_stale_by*, *t_threshold*)

raise_initial_state()

raise_no_next_check_log_entry()

raise_notification_log_entry(*n*)

reactionner_tag

resultmodulations

retain_nonstatus_information

retain_status_information

retry_interval

return_code

running_properties = {'state_id_before_impact': <shinken.property.IntegerProp object at 0x7fa5c3d59e50>, 'child_

scheduled_downtime_depth

service_dependencies

service_description

servicegroups

set_impact_state()

set_state_from_exit_status(*status*)

set_unreachable()

should_be_scheduled

source_problems

stalking_options

start_time

state

state_before_hard_unknown_reach_phase

state_before_impact

state_changed_since_impact

state_id

state_id_before_impact

state_type

state_type_id

tags

time_to_orphanage

timeout

topology_change

trending_policies

trigger

trigger_name

triggers

unset_impact_state()

use

was_in_hard_unknown_reach_phase

class shinken.objects.service.**Services**(*items*)

Bases: *shinken.objects.item.Items*

apply_dependencies()

apply_implicit_inheritance(*hosts*)

apply_inheritance(*hosts*)

copy_create_service_from_another(*hosts*, *s*, *hname*)

create_business_rules(*hosts*, *services*)

create_business_rules_dependencies()

create_reversed_list()

delete_services_by_id(*ids*)

explode (*hosts*, *hostgroups*, *contactgroups*, *servicegroups*, *servicedependencies*, *triggers*)

find_srv_by_name_and_hostname (*host_name*, *name*)

find_srv_id_by_name_and_hostname (*host_name*, *name*)

inner_class
alias of *Service*

late_linkify_s_by_commands (*commands*)

linkify (*hosts*, *commands*, *timeperiods*, *contacts*, *resultmodulations*, *businessimpactmodulations*, *escalations*, *servicegroups*, *triggers*, *checkmodulations*, *macromodulations*)

linkify_s_by_hst (*hosts*)

linkify_s_by_sg (*servicegroups*)

optimize_service_search (*hosts*)

servicedependency Module

class shinken.objects.servicedependency.**Servicedependencies** (*items*)
Bases: *shinken.objects.item.Items*

add_service_dependency (*dep_host_name*, *dep_service_description*, *par_host_name*, *par_service_description*)

apply_inheritance (*hosts*)

delete_servicesdep_by_id (*ids*)

explode (*hostgroups*)

explode_hostgroup (*sd*, *hostgroups*)

linkify (*hosts*, *services*, *timeperiods*)

linkify_s_by_sd ()

linkify_sd_by_s (*hosts*, *services*)

linkify_sd_by_tp (*timeperiods*)

class shinken.objects.servicedependency.**Servicedependency** (*params={}*)
Bases: *shinken.objects.item.Item*

get_name ()

id = 0

my_type = 'servicedependency'

properties = {'inherits_parent': <shinken.property.BoolProp object at 0x7fa5c40c4490>, 'use': <shinken.property.List

serviceescalation Module

class shinken.objects.serviceescalation.**Serviceescalation** (*params={}*)
Bases: *shinken.objects.item.Item*

get_name ()

id = 1

my_type = 'serviceescalation'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

class shinken.objects.serviceescalation.**Serviceescalations**(*items*)
Bases: *shinken.objects.item.Items*

explode(*escalations*)

inner_class
alias of *Serviceescalation*

name_property = ''

serviceextinfo Module This is the main class for the Service ext info. In fact it's mainly about the configuration part. Parameters are merged in Service so it's no use in running part

class shinken.objects.serviceextinfo.**ServiceExtInfo**(*params={}*)
Bases: *shinken.objects.item.Item*

get_dbg_name()

get_full_name()

get_name()

host_name

icon_image

icon_image_alt

id = 1

imported_from

is_correct()

macros = {'SERVICENOTES': 'notes', 'SERVICEDESC': 'service_description', 'SERVICENOTESURL': 'notes_url', '

my_type = 'serviceextinfo'

name

notes

notes_url

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

service_description

use

class shinken.objects.serviceextinfo.**ServicesExtInfo**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *ServiceExtInfo*

merge(*services*)

merge_extinfo(*service*, *extinfo*)

name_property = 'host_name'

servicegroup Module

class shinken.objects.servicegroup.**Servicegroup**(*params={}*)

Bases: *shinken.objects.itemgroup.Itemgroup*

get_name()

get_servicegroup_members()

get_services()

get_services_by_explosion(*servicegroups*)

id = 1

macros = {'SERVICEGROUPNOTES': 'notes', 'SERVICEGROUPALIAS': 'alias', 'SERVICEGROUPMEMBERS': 'm

my_type = 'servicegroup'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

class shinken.objects.servicegroup.**Servicegroups**(*items*)

Bases: *shinken.objects.itemgroup.Itemgroups*

add_member(*cname*, *sgname*)

explode()

inner_class

alias of *Servicegroup*

linkify(*services*)

linkify_sg_by_srv(*services*)

name_property = 'servicegroup_name'

timeperiod Module

class shinken.objects.timeperiod.**Timeperiod**(*params={}*)

Bases: *shinken.objects.item.Item*

apply_inheritance()

check_and_log_activation_change()

check_exclude_rec()

clean_cache()

explode(*timeperiods*)

fill_data_brok_from(*data*, *brok_type*)

find_next_invalid_time_from_cache(*t*)

find_next_valid_time_from_cache(*t*)

get_initial_status_brok()

get_min_from_t(*t*)

get_name()

get_next_invalid_time_from_t(*t*)

get_next_valid_time_from_t(*t*)

get_not_in_min_from_t(*f*)

get_raw_import_values()

get_unresolved_properties_by_inheritance(*items*)

has(*prop*)

id = 1

is_correct()

is_time_valid(*t*)

linkify(*timeperiods*)

my_type = 'timeperiod'

properties = {'use': <shinken.property.StringProp object at 0x7fa5c3ee05d0>, 'name': <shinken.property.StringProp

resolve_daterange(*dateranges*, *entry*)

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.timeperiod.**Timeperiods**(*items*)

Bases: *shinken.objects.item.Items*

apply_inheritance()

explode()

inner_class

alias of *Timeperiod*

is_correct()

linkify()

name_property = 'timeperiod_name'

trigger Module

class shinken.objects.trigger.**Trigger**(*params={}*)

Bases: *shinken.objects.item.Item*

compile()

eval(*myself*, *ctx*)

get_name()

id = 1

my_type = 'trigger'

properties = {'trigger_name': <shinken.property.StringProp object at 0x7fa5c5d36510>, 'use': <shinken.property.List

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.trigger.**Triggers**(*items*)

Bases: *shinken.objects.item.Items*

compile()

create_trigger(*src*, *name*)

inner_class

alias of *Trigger*

load_file(*path*)

load_objects(*conf*)

name_property = 'trigger_name'

webui Package

webui Package

bottlecore Module

bottlewebui Module

17.2 shinken Package

17.2.1 shinken Package

17.2.2 acknowledge Module

class shinken.acknowledge.**Acknowledge**(*ref*, *sticky*, *notify*, *persistent*, *author*, *comment*, *end_time=0*)

Allows you to acknowledge the current problem for the specified service. By acknowledging the current problem, future notifications (for the same servicestate) are disabled.

id = 1

properties = {'comment': None, 'author': None, 'sticky': None, 'end_time': None, 'id': None, 'notify': None}

17.2.3 action Module

class shinken.action.**Action**

Bases: shinken.action.__Action

execute__(*force_shell=False*)

kill__()

shinken.action.**no_block_read**(*output*)

17.2.4 arbiterlink Module

17.2.5 autoslots Module

The AutoSlots Class is a MetaClass: it manages how other classes are created (Classes, not instances of theses classes). Here it's role is to create the __slots__ list of the class with all properties of Class.properties and Class.running_properties so we do not have to add manually all properties to the __slots__ list when we add a new entry

class shinken.autoslots.**AutoSlots**

Bases: type

17.2.6 basemodule Module

This python module contains the class BaseModule that shinken modules will subclass

class shinken.basemodule.**BaseModule**(*mod_conf*)

Bases: object

This is the base class for the shinken modules. Modules can be used by the different shinken daemons/services for different tasks. Example of task that a shinken module can do:

- load additional configuration objects.
- recurrently save hosts/services status/perfdata informations in different format.
- ...

clear_queues(*manager*)

Release the resources associated to the queues of this instance

create_queues(*manager=None*)

The manager is None on android, but a true Manager() elsewhere Create the shared queues that will be used by shinken daemon process and this module process. But clear queues if they were already set before recreating new one.

do_loop_turn()

For external modules only: implement in this method the body of you main loop

do_stop()

Called just before the module will exit Put in this method all you need to cleanly release all open resources used by your module

get_name()

has(*prop*)

The classic has: do we have a prop or not?

init()

Handle this module "post" init ; just before it'll be started. Like just open necessaries file(s), database(s), or whatever the module will need.

main()

module "main" method. Only used by external modules.

manage_brok(*brok*)

Request the module to manage the given brok. There a lot of different possible broks to manage.

manage_signal(*sig*, *frame*)

set_exit_handler(*sigs=None*)

set_loaded_into(*daemon_name*)

set_proctitle(*name*)

set_signal_handler(*sigs=None*)

start()

stop_process()

Request the module process to stop and release it

want_brok(*b*)

work()

module "main" method. Only used by external modules.

class shinken.basemodule.**ModulePhases**

TODO: Add some comment about this class for the doc

CONFIGURATION = 1

LATE_CONFIGURATION = 2

RETENTION = 8

RUNNING = 4

17.2.7 bin Module

17.2.8 borg Module

class shinken.borg.**Borg**

Bases: object

Here is the new-style Borg (not much more complex then the "old-style")

17.2.9 brok Module

class shinken.brok.**Brok**(*type*, *data*)

A Brok is a piece of information exported by Shinken to the Broker. Broker can do whatever he wants with it.

id = 0

my_type = 'brok'

prepare()

17.2.10 brokerlink Module

17.2.11 check Module

class shinken.check.**Check**(*status*, *command*, *ref*, *t_to_go*, *dep_check=None*, *id=None*, *timeout=10*, *poller_tag='None'*, *reactionner_tag='None'*, *env={}*, *module_type='fork'*, *from_trigger=False*, *dependency_check=False*)

Bases: *shinken.action.Action*

ODO: Add some comment about this class for the doc

check_time

check_type

copy_shell()

return a copy of the check but just what is important for execution So we remove the ref and all

dep_check

depend_on

env

execution_time

exit_status

from_trigger

get_id()

get_return_from(*c*)

internal

is_a

is_dependent()

is_launchable(*t*)

long_output

module_type

my_type = 'check'

output

perf_data

poller_tag

properties = {'check_type': <shinken.property.IntegerProp object at 0x7fa5c5c8a710>, 'exit_status': <shinken.proper

reactionner_tag

ref

s_time

set_type_active()

set_type_passive()

state

status

t_to_go

type

u_time

worker

17.2.12 commandcall Module

class shinken.commandcall.**CommandCall**(*commands*, *call*, *poller_tag='None'*, *reactionner_tag='None'*)

Bases: *shinken.commandcall.DummyCommandCall*

This class is use when a service, contact or host define a command with args.

args

call

command

get_command_and_args()

We want to get the command and the args with ! splitting. but don't forget to protect against the ! to do not split them

get_name()

id = 0

is_valid()

late_linkify_with_command(*commands*)

late_relink_done

module_type

my_type = 'CommandCall'

poller_tag

properties = {'poller_tag': <shinken.property.StringProp object at 0x7fa5c5ba0b10>, 'reactionner_tag': <shinken.pro

reactionner_tag

timeout

valid

class shinken.commandcall.**DummyCommandCall**

Bases: object

Ok, slots are fun: you cannot set the __autoslots__ on the same class you use, fun isn't it? So we define* a dummy useless class to get such :)

17.2.13 comment Module

class shinken.comment.**Comment**(*ref*, *persistent*, *author*, *comment*, *comment_type*, *entry_type*, *source*, *expires*, *expire_time*)

id = 1

properties = {'comment': None, 'source': None, 'entry_time': None, 'comment_type': None, 'author': None, 'expire_

17.2.14 complexexpression Module

class shinken.complexexpression.**ComplexExpressionFactory**(*ctx='hostgroups'*, *grps=None*, *all_elements=None*)

Bases: object

eval_cor_pattern(*pattern*)

find_object(*pattern*)

class shinken.complexexpression.**ComplexExpressionNode**

Bases: object

is_valid()

resolve_elements()

17.2.15 contactdowntime Module

class shinken.contactdowntime.**ContactDowntime**(*ref*, *start_time*, *end_time*, *author*, *comment*)

cancel()

check_activation()

enter()

exit()

id = 1

in_scheduled_downtime()

properties = {'comment': None, 'author': None, 'start_time': None, 'is_in_effect': None, 'end_time': None, 'can_be_c

17.2.16 daemon Module

17.2.17 daterange Module

class shinken.daterange.**CalendarDaterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

class shinken.daterange.**Daterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

get_min_from_t(*t*)

get_min_sec_from_morning()

get_min_sec_out_from_morning()

classmethod get_month_by_id(*id*)

classmethod get_month_id(*month*)

get_next_future_timerange_invalid(*t*)

get_next_future_timerange_valid(*t*)

get_next_invalid_day(*t*)

get_next_invalid_time_from_t(*t*)

get_next_valid_day(*t*)

get_next_valid_time_from_t(*t*)

get_start_and_end_time(*ref=None*)

classmethod get_weekday_by_id(*id*)

classmethod get_weekday_id(*weekday*)

is_correct()

is_time_day_invalid(*t*)

is_time_day_valid(*t*)

is_time_valid(*t*)

months = {'march': 3, 'august': 8, 'september': 9, 'december': 12, 'june': 6, 'july': 7, 'november': 11, 'february': 2, 'oct

weekdays = {'monday': 0, 'tuesday': 1, 'friday': 4, 'wednesday': 2, 'thursday': 3, 'sunday': 6, 'saturday': 5}

class shinken.daterange.**MonthDateDaterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

class shinken.daterange.**MonthDayDaterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

class shinken.daterange.**MonthWeekDayDaterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

is_correct()

class shinken.daterange.**StandardDaterange**(*day*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

is_correct()

class shinken.daterange.**Timerange**(*entry*)

get_first_sec_out_from_morning()

get_sec_from_morning()

is_time_valid(*t*)

class shinken.daterange.**WeekDayDaterange**(*syear*, *smon*, *smday*, *swday*, *swday_offset*, *eyear*, *emon*, *emday*, *ewday*, *ewday_offset*, *skip_interval*, *other*)

Bases: *shinken.daterange.Daterange*

get_start_and_end_time(*ref=None*)

shinken.daterange.**find_day_by_offset**(*year*, *month*, *offset*)

shinken.daterange.**find_day_by_weekday_offset**(*year*, *month*, *weekday*, *offset*)

17.2.18 db Module

class shinken.db.**DB**(*table_prefix=''*)

Bases: object

DB is a generic class for SQL Database

create_insert_query(*table*, *data*)

Create a INSERT query in table with all data of data (a dict)

create_update_query(*table*, *data*, *where_data*)

Create a update query of table with data, and use where data for the WHERE clause

fetchone()

Just get an entry

stringify(*val*)

Get a unicode from a value

17.2.19 db_mysql Module

class shinken.db_mysql.**DBMysql**(*host*, *user*, *password*, *database*, *character_set*, *table_prefix=''*, *port=3306*)

Bases: *shinken.db.DB*

DBMysql is a MySQL access database class

connect_database()

Create the database connection TODO: finish (begin :)) error catch and conf parameters... Import to catch exception

execute_query(*query*, *do_debug=False*)

Just run the query TODO: finish catch

17.2.20 db_oracle Module

class shinken.db_oracle.**DBOracle**(*user*, *password*, *database*, *table_prefix=''*)

Bases: *shinken.db.DB*

Manage connection and query execution against Oracle databases.

connect_database()

Create the database connection TODO: finish (begin :)) error catch and conf parameters...

execute_query(*query*)

Execute a query against an Oracle database.

17.2.21 db_sqlite Module

class shinken.db_sqlite.**DBSqlite**(*db_path*, *table_prefix=''*)

Bases: *shinken.db.DB*

DBSqlite is a sqlite access database class

connect_database()

Create the database connection

execute_query(*query*)

Just run the query

17.2.22 dependencynode Module

class shinken.dependencynode.**DependencyNode**

Bases: object

get_reverse_state(*state*)

get_state()

is_valid()

list_all_elements()

switch_zeros_of_values()

class shinken.dependencynode.**DependencyNodeFactory**

Bases: object

eval_cor_pattern(*pattern*, *hosts*, *services*)

find_object(*pattern*, *hosts*, *services*)

17.2.23 dispatcher Module

This is the class of the dispatcher. Its role is to dispatch configurations to other elements like schedulers, reactionner, pollers, receivers and brokers. It is responsible for high availability part. If an element dies and the element type has a spare, it sends the config of the dead one to the spare

class shinken.dispatcher.**Dispatcher**(*conf*, *arbiter*)

check_alive()

check_bad_dispatch()

check_dispatch()

dispatch()

get_scheduler_ordered_list(*r*)

17.2.24 downtime Module

class shinken.downtime.**Downtime**(*ref*, *start_time*, *end_time*, *fixed*, *trigger_id*, *duration*, *author*, *comment*)

add_automatic_comment()

cancel()

del_automatic_comment()

enter()

exit()

fill_data_brok_from(*data*, *brok_type*)

get_initial_status_brok()

id = 1

in_scheduled_downtime()

properties = {'comment': <shinken.property.StringProp object at 0x7fa5c44a7550>, 'entry_time': <shinken.property.

trigger_me(*other_downtime*)

17.2.25 easter Module

shinken.easter.**dark**()

.-.

|_:_|

/(_Y_)

(/M/)

'. _.'-/'-'-'._

```
            ': _/.–'[[[[]'–._
                ': /_'  [|::"|][' .\]
                            ': // ./ |oUU| .' :
                                ': _:'..' \_|___|_/  [:|]
                                        ':. .' |_[____]_| :.':
                                          [::\ |  [| |][;][\] '-' /'.| |.'  .;.' | |_ '-' : | | .: : | | | | '. : | / :. .; |
                                          / | | :__/ : \

                                      | | : | | ||

                                  / : : |: / |__| /|

                        snd |  [: :_/_| /'._\ '–|_]
                            /___.-/_|-'   '-'
shinken.easter.episode_iv()
shinken.easter.get_coffee()
                                      (
                                        ) (  ____...(———-)-....___
                                  .-"" ) ( ""-.
                                .-'''|-._ ) _.-|
                            / .–.| " "—...........—" " |
                        / / | | | | | |
                            | |
                                | |
                                      ''| | _/ / /
                                  (__/ /
                        _..—""' /''""—.._
                    .-' / '-.
            : '-.__ __.-' : : ) ""—...—"" ( :
                '._ "–...____...–" _.'
        jgs ""–.._ __..–""/
            '._ ""—-.....______.....—-""  _.'
                ""–..,,_____ _____,,..–""  " " '__- " " '
shinken.easter.myip()
shinken.easter.naheulbeuk()
shinken.easter.perdu()
```

17.2.26 eventhandler Module

class shinken.eventhandler.**EventHandler**(*command*, *id=None*, *ref=None*, *timeout=10*, *env={}*, *module_type='fork'*, *reactionner_tag='None'*)

Bases: *shinken.action.Action*

check_time

command

copy_shell()

env

execution_time

exit_status

get_id()

get_outputs(*out*, *max_plugins_output_length*)

get_return_from(*e*)

is_a

is_launchable(*t*)

long_output

module_type

my_type = 'eventhandler'

output

perf_data

properties = {'status': <shinken.property.StringProp object at 0x7fa5c42ece50>, 'module_type': <shinken.property.St

reactionner_tag

s_time

sched_id

status

t_to_go

timeout

type

u_time

worker

17.2.27 external_command Module

17.2.28 graph Module

class shinken.graph.**Graph**

Graph is a class to make graph things like DFS checks or accessibility Why use an atomic bomb when a little hammer is enough?

add_edge(*from_node*, *to_node*)

add_node(*node*)

add_nodes(*nodes*)

dfs_get_all_childs(*root*)

dfs_loop_search(*root*)

get_accessibility_packs()

loop_check()

17.2.29 `http_client` Module

17.2.30 `http_daemon` Module

17.2.31 `load` Module

class `shinken.load.`**Load**(*m=1*, *initial_value=0*)

This class is for having a easy Load calculation without having to send value at regular interval (but it's more efficient if you do this :)) and without having a list or other stuff. It's just an object, an update and a get You can define m: the average for m minutes. The val is the initial value. It's better if it's 0 but you can choose.

get_load()

update_load(*new_val*, *forced_interval=None*)

17.2.32 `log` Module

class `shinken.log.`**Log**

Shinken logger class, wrapping access to Python logging standard library.

CRITICAL = 50

DEBUG = 10

ERROR = 40

INFO = 20

NOTSET = 0

WARNING = 30

critical(*msg*, **args*, ***kwargs*)

debug(*msg*, **args*, ***kwargs*)

error(*msg*, **args*, ***kwargs*)

get_level()

static get_level_id(*lvlName*)

Convert a level name (string) to its integer value and vice-versa. Input a level and it will return a name. Raise KeyError when name or level not found

info(*msg*, **args*, ***kwargs*)

load_obj(*object*, *name_=None*)

We load the object where we will put log broks with the 'add' method

log (*message*, *format=None*, *print_it=True*)
Old log method, kept for NAGIOS compatibility What strings should not use the new format ??

quit ()
Close the local log file at program exit

register_local_log (*path*, *level=None*)
The shinken logging wrapper can write to a local file if needed and return the file descriptor so we can avoid to close it.

set_human_format (*on=True*)
Set the output as human format.

If the optional parameter *on* is False, the timestamps format will be reset to the default format.

set_level (*level*)

warning (*msg*, **args*, ***kwargs*)

shinken.log.**cprint** (*s*, *color*)

shinken.log.**is_tty** ()

17.2.33 macroresolver Module

class shinken.macroresolver.**MacroResolver**
Bases: *shinken.borg.Borg*

Please Add a Docstring to describe the class here

get_env_macros (*data*)

init (*conf*)

macros = {'TOTALSERVICESCRITICALUNHANDLED': '_get_total_services_critical_unhandled', 'PROCESSSTART

my_type = 'macroresolver'

resolve_command (*com*, *data*)

resolve_simple_macros_in_string (*c_line*, *data*, *args=None*)

17.2.34 memoized Module

class shinken.memoized.**memoized** (*func*)
Bases: object

Decorator that caches a function's return value each time it is called. If called later with the same arguments, the cached value is returned, and not re-evaluated.

17.2.35 message Module

class shinken.message.**Message** (*id*, *type*, *data=None*, *source=None*)
This is a simple message class for communications between actionners and workers

get_data ()

get_from ()

get_type ()

my_type = 'message'

str()

17.2.36 modulesctx Module

17.2.37 modulesmanager Module

class shinken.modulesmanager.**ModulesManager**(*modules_type*, *modules_path*, *modules*)

Bases: object

This class is use to manage modules and call callback

check_alive_instances()

clear_instances(*insts=None*)

get_external_from_queues()

get_external_instances(*phase=None*)

get_external_to_queues()

get_instances()

get_internal_instances(*phase=None*)

load()

load_and_init()

load_manager(*manager*)

remove_instance(*inst*)

set_max_queue_size(*max_queue_size*)

set_modules(*modules*)

set_to_restart(*inst*)

start_external_instances(*late_start=False*)

stop_all()

try_instance_init(*inst*, *late_start=False*)

try_to_restart_deads()

17.2.38 notification Module

class shinken.notification.**Notification**(*type='PROBLEM'*, *status='scheduled'*, *command='UNSET'*, *command_call=None*, *ref=None*, *contact=None*, *t_to_go=0*, *contact_name=''*, *host_name=''*, *service_description=''*, *reason_type=1*, *state=0*, *ack_author=''*, *ack_data=''*, *escalated=False*, *contacts_notified=0*, *start_time=0*, *end_time=0*, *notification_type=0*, *id=None*, *notif_nb=1*, *timeout=10*, *env={}*, *module_type='fork'*, *reactionner_tag='None'*)

Bases: shinken.action.Action

Please Add a Docstring to describe the class here

ack_author

ack_data

already_start_escalations

check_time

command

command_call

contact

contact_name

contacts_notified

copy_shell()

creation_time

end_time

env

escalated

execution_time

exit_status

fill_data_brok_from(*data*, *brok_type*)

get_id()

get_initial_status_brok()

get_return_from(*n*)

host_name

is_a

is_administrative()

is_launchable(*t*)

macros = {'NOTIFICATIONAUTHORNAME': 'author_name', 'HOSTNOTIFICATIONNUMBER': 'notif_nb', 'NOTI

module_type

my_type = 'notification'

notif_nb

notification_type

output

properties = {'reason_type': <shinken.property.StringProp object at 0x7fa5c3e16dd0>, 'status': <shinken.property.St

reactionner_tag

reason_type

s_time

sched_id

service_description

start_time

state

status

t_to_go

timeout

type

u_time

worker

17.2.39 pollerlink Module

17.2.40 property Module

class shinken.property.**UnusedProp**(*text=None*)

Bases: shinken.property.Property

A unused Property. These are typically used by Nagios but no longer useful/used by Shinken.

This is just to warn the user that the option he uses is no more used in Shinken.

class shinken.property.**BoolProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

A Boolean Property.

Boolean values are currently case insensitively defined as 0, false, no, off for False, and 1, true, yes, on for True).

pythonize(*val*)

class shinken.property.**IntegerProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

Please Add a Docstring to describe the class here

pythonize(*val*)

class shinken.property.**FloatProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

Please Add a Docstring to describe the class here

pythonize(*val*)

class shinken.property.**CharProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

Please Add a Docstring to describe the class here

pythonize(*val*)

class shinken.property.**StringProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

Please Add a Docstring to describe the class here

pythonize(*val*)

class shinken.property.**ListProp**(*default=<object object>, class_inherit=None, unmanaged=False, help='', no_slots=False, fill_brok=None, conf_send_preparation=None, brok_transformation=None, retention=False, retention_preparation=None, to_send=False, override=False, managed=True*)

Bases: shinken.property.Property

Please Add a Docstring to describe the class here

pythonize(*val*)

17.2.41 reactionnerlink Module

17.2.42 receiverlink Module

17.2.43 satellite Module

17.2.44 satellitelink Module

17.2.45 scheduler Module

17.2.46 schedulerlink Module

17.2.47 singleton Module

class shinken.singleton.**Singleton**(*name, bases, dict*)

Bases: type

The classic Singleton class. So all instance of this class will be the same instance in fact.

17.2.48 sorteddict Module

class shinken.sorteddict.**SortedDict**(**args, **kwargs*)

Bases: dict

Dictionary with sorted keys.

The interface is similar to the ordinary dict's one, but: * methods: __repr__(), __str__(), __iter__(), iterkeys(), itervalues(),
iteritems(), keys(), values(), items() and popitem() – return results taking into consideration sorted keys order;

•new methods: largest_key(), largest_item(), smallest_key(), smallest_item() added.

clear() → None. Remove all items from D.

copy() → a shallow copy of D (still as a SortedDict).

classmethod fromkeys(*S*[, *v*]) → New dict with keys from S and values equal to v.
v defaults to None.

items() → a list of (key, value) pairs sorted by keys
(add reverse=True for reverse ordering).

iteritems() → an iterator over (key, value) pairs sorted by keys
(add reverse=True for reverse ordering).

iterkeys(*reverse=False*)
D.__iter__() <==> iter(D) <==> D.iterkeys() -> an iterator over sorted keys (add reverse=True for reverse ordering).

itervalues() → an iterator over values sorted by keys
(add reverse=True for reverse ordering).

keys() → a sorted list of keys
(add reverse=True for reverse ordering).

largest_item() → a (key, value) pair with the largest key;
raise KeyError if D is empty.

largest_key() → the largest key; raise KeyError if D is empty.

pop(*k*[, *d*]) → v, remove specified key and return the corresponding value.
If key is not found, d is returned if given, otherwise KeyError is raised

popitem() → (k, v). Remove and return a (key, value) pair with
the largest key; raise KeyError if D is empty.

setdefault(*k*[, *d*]) → D.get(k,d), also set D[k]=d if k not in D

smallest_item() → a (key, value) pair with the smallest key;
raise KeyError if D is empty.

smallest_key() → the smallest key; raise KeyError if D is empty.

update([*E*], **F) → None. Update D from dict/iterable E and F.
If E present and has a .keys() method, does: for k in E: D[k] = E[k] If E present and lacks .keys() method, does: for (k, v) in E: D[k] = v In either case, this is followed by: for k in F: D[k] = F[k]

values() → a list of values sorted by keys
(add reverse=True for reverse ordering).

shinken.sorteddict.**dictdoc**(*method*)
A decorator making reuse of the ordinary dict's docstrings more concise.

17.2.49 `trigger_functions` Module

17.2.50 `util` Module

shinken.util.**alive_then_spare_then_deads**(*x*, *y*)

shinken.util.**apply_change_recursive_pattern_change**(*s*, *rule*)

shinken.util.**expand_with_macros**(*ref*, *value*)

shinken.util.**expand_xy_pattern**(*pattern*)

shinken.util.**expect_file_dirs**(*root*, *path*)

shinken.util.**format_t_into_dhms_format**(*t*)

shinken.util.**from_bool_to_int**(*b*)

shinken.util.**from_bool_to_string**(*b*)

shinken.util.**from_float_to_int**(*val*)

shinken.util.**from_list_to_split**(*val*)

shinken.util.**get_customs_keys**(*d*)

shinken.util.**get_customs_values**(*d*)

shinken.util.**get_day**(*t*)

shinken.util.**get_end_of_day**(*year*, *month_id*, *day*)

shinken.util.**get_key_value_sequence**(*entry*, *default_value=None*)

shinken.util.**get_obj_full_name**(*obj*)

shinken.util.**get_obj_name**(*obj*)

shinken.util.**get_obj_name_two_args_and_void**(*obj*, *value*)

shinken.util.**get_sec_from_morning**(*t*)

shinken.util.**get_start_of_day**(*year*, *month_id*, *day*)

shinken.util.**get_wday**(*t*)

shinken.util.**got_generation_rule_pattern_change**(*xy_couples*)

shinken.util.**nighty_five_percent**(*t*)

shinken.util.**print_date**(*t*)

shinken.util.**safe_print**(**args*)

shinken.util.**scheduler_no_spare_first**(*x*, *y*)

shinken.util.**sort_by_ids**(*x*, *y*)

shinken.util.**strip_and_uniq**(*tab*)

shinken.util.**to_best_int_float**(*val*)

shinken.util.**to_bool**(*val*)

shinken.util.**to_char**(*val*)

shinken.util.**to_float**(*val*)

shinken.util.**to_hostnames_list**(*ref*, *tab*)

shinken.util.**to_int**(*val*)

shinken.util.**to_list_of_names**(*ref*, *tab*)

shinken.util.**to_list_string_of_names**(*ref*, *tab*)

shinken.util.**to_name_if_possible**(*ref*, *value*)

shinken.util.**to_split**(*val*)

shinken.util.**to_svc_hst_distinct_lists**(*ref*, *tab*)

17.2.51 worker Module

class shinken.worker.**Worker**(*id*, *s*, *returns_queue*, *processes_by_worker*, *mortal=True*, *timeout=300*, *max_plugins_output_length=8192*, *target=None*, *loaded_into='unknown'*)

This class is used for poller and reactionner to work. The worker is a process launch by theses process and read Message in a Queue (self.s) (slave) They launch the Check and then send the result in the Queue self.m (master) they can die if they do not do anything (param timeout)

add_idletime(*time*)

check_for_system_time_change()

do_work(*s*, *returns_queue*, *c*)

get_new_checks()

id = **0**

is_alive()

is_killable()

is_mortal()

join(*timeout=None*)

launch_new_checks()

manage_finished_checks()

reset_idle()

send_message(*msg*)

set_proctitle()

set_zombie()

start()

terminate()

work(*s*, *returns_queue*, *c*)

17.2.52 Subpackages

clients Package

clients Package

LSB Module

class shinken.clients.LSB.**FileQueryGenerator**(*filename*)
Bases: *shinken.clients.LSB.SimpleQueryGenerator*

class shinken.clients.LSB.**QueryGenerator**
Bases: object

Generate a livestatus query

get()

class shinken.clients.LSB.**SimpleQueryGenerator**(*querys*, *name='sqg'*)
Bases: *shinken.clients.LSB.QueryGenerator*

get()

shinken.clients.LSB.**main**(*argv*)

shinken.clients.LSB.**mean**(*numberList*)

shinken.clients.LSB.**median**(*numberList*)

shinken.clients.LSB.**run**(*url*, *requests*, *concurrency*, *qg*)

shinken.clients.LSB.**usage**()

livestatus Module

class shinken.clients.livestatus.**LSAsynConnection**(*addr='127.0.0.1'*, *port=50000*, *path=None*, *timeout=10*)
Bases: asyncore.dispatcher

do_connect()

do_read(*size*)

exec_command(*command*)

get_query()

get_returns()

handle_close()

handle_connect()

handle_read()

handle_write()

is_finished()

launch_raw_query(*query*)

look_for_timeout()

readable()

stack_query(*q*)

wait_returns()

writable()

class shinken.clients.livestatus.**LSConnectionPool**(*con_addrs*)
Bases: object

launch_raw_query(*query*)

class shinken.clients.livestatus.**LSSyncConnection**(*addr='127.0.0.1'*, *port=50000*, *path=None*, *timeout=10*)

connect()

exec_command(*command*)

launch_query(*query*)

read(*size*)

class shinken.clients.livestatus.**Query**(*q*)
Bases: object

get()

id = 0

put(*r*)

daemons Package

daemons Package

arbiterdaemon Module

brokerdaemon Module

pollerdaemon Module

reactionnerdaemon Module

receiverdaemon Module

schedulerdaemon Module

discovery Package

discovery Package

discoverymanager Module

misc Package

misc Package

datamanager Module

class shinken.misc.datamanager.**DataManager**

Bases: object

get_all_hosts_and_services()

get_all_impacts()

get_all_problems(*to_sort=True*, *get_acknowledged=False*)

get_brokers()

get_business_parents(*obj*, *levels=3*)

get_contact(*name*)

get_contacts()

get_host(*hname*)

get_host_tags_sorted()

get_hostgroup(*name*)

get_hostgroups()

get_hostgroups_sorted(*selected=''*)

get_hosts()

get_important_elements()

get_important_impacts()

get_len_overall_state()

get_nb_all_problems()

get_nb_elements()

get_nb_impacts()

get_nb_problems()

get_overall_it_state()

get_overall_state()

get_per_hosts_state()

get_per_service_state()

get_pollers()

get_problems_time_sorted()

get_program_start()

get_reactionners()

get_realm(*r*)

get_realms()

get_receivers()

get_schedulers()

get_service(*hname*, *sdesc*)

get_services()

guess_root_problems(*obj*)

load(*rg*)

filter Module

Helper functions for some filtering, like for user based

shinken.misc.filter.**only_related_to**(*lst*, *user*)

md5crypt Module

md5crypt.py - Provides interoperable MD5-based crypt() function

SYNOPSIS

import md5crypt.py

cryptedpassword = md5crypt.md5crypt(password, salt);

DESCRIPTION

unix_md5_crypt() provides a crypt()-compatible interface to the rather new MD5-based crypt() function found in modern operating systems. It's based on the implementation found on FreeBSD 2.2.[56]-RELEASE and contains the following license in it:

> "THE BEER-WARE LICENSE" (Revision 42): <phk@login.dknet.dk> wrote this file. As long as you retain this notice you can do whatever you want with this stuff. If we meet some day, and you think this stuff is worth it, you can buy me a beer in return. Poul-Henning Kamp

apache_md5_crypt() provides a function compatible with Apache's .htpasswd files. This was contributed by Bryan Hart <bryan@eai.com>.

`shinken.misc.md5crypt.`**`apache_md5_crypt`**(*pw*, *salt*)

`shinken.misc.md5crypt.`**`md5crypt`**(*pw*, *salt*, *magic=None*)

`shinken.misc.md5crypt.`**`to64`**(*v*, *n*)

`shinken.misc.md5crypt.`**`unix_md5_crypt`**(*pw*, *salt*, *magic=None*)

`perfdata` Module

class `shinken.misc.perfdata.`**`Metric`**(*s*)

class `shinken.misc.perfdata.`**`PerfDatas`**(*s*)

`shinken.misc.perfdata.`**`guess_int_or_float`**(*val*)

`regenerator` Module

`sorter` Module

Helper functions for some sorting

`shinken.misc.sorter.`**`hst_srv_sort`**(*s1*, *s2*)

`shinken.misc.sorter.`**`last_state_change_earlier`**(*s1*, *s2*)

`shinken.misc.sorter.`**`worse_first`**(*s1*, *s2*)

`termcolor` Module

ANSII Color formatting for output in terminal.

`shinken.misc.termcolor.`**`colored`**(*text*, *color=None*, *on_color=None*, *attrs=None*)

Colorize text.
Available text colors: red, green, yellow, blue, magenta, cyan, white.
Available text highlights: on_red, on_green, on_yellow, on_blue, on_magenta, on_cyan, on_white.
Available attributes: bold, dark, underline, blink, reverse, concealed.
Example: colored('Hello, World!', 'red', 'on_grey', ['blue', 'blink']) colored('Hello, World!', 'green')

`shinken.misc.termcolor.`**`cprint`**(*text*, *color=None*, *on_color=None*, *attrs=None*, ***kwargs*)

Print colorize text.

It accepts arguments of print function.

objects Package

`objects` Package

The objects package contains definition classes of the different objects that can be declared in configuration files.

businessimpactmodulation Module

class shinken.objects.businessimpactmodulation.**Businessimpactmodulation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

my_type = 'businessimpactmodulation'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

class shinken.objects.businessimpactmodulation.**Businessimpactmodulations**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *Businessimpactmodulation*

linkify(*timeperiods*)

linkify_cm_by_tp(*timeperiods*)

name_property = 'business_impact_modulation_name'

checkmodulation Module

class shinken.objects.checkmodulation.**CheckModulation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_check_command(*t_to_go*)

get_name()

id = 1

is_correct()

late_linkify_cw_by_commands(*commands*)

macros = {}

my_type = 'checkmodulation'

properties = {'check_period': <shinken.property.StringProp object at 0x7fa5c436d490>, 'use': <shinken.property.List

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.checkmodulation.**CheckModulations**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *CheckModulation*

linkify(*timeperiods*, *commands*)

name_property = 'checkmodulation_name'

new_inner_member(*name=None*, *params={}*)

command Module

class shinken.objects.command.**Command**(*params={}*)
Bases: *shinken.objects.item.Item*

command_line

command_name

fill_data_brok_from(*data*, *brok_type*)

get_initial_status_brok()

get_name()

id = **0**

imported_from

module_type

my_type = **'command'**

name

poller_tag

properties = **{'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob**

pythonize()

reactionner_tag

timeout

use

class shinken.objects.command.**Commands**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *Command*

name_property = **'command_name'**

class shinken.objects.command.**DummyCommand**
Bases: object

config Module

contact Module

class shinken.objects.contact.**Contact**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

get_notification_commands(*type*)

id = **1**

is_correct()

macros = **{'CONTACTPAGER': 'pager', 'CONTACTGROUPNAMES': 'get_groupnames', 'CONTACTGROUPNAME'**

my_type = 'contact'

old_properties = {'min_criticity': 'min_business_impact'}

properties = {'address1': <shinken.property.StringProp object at 0x7fa5c40b33d0>, 'address2': <shinken.property.St

raise_cancel_downtime_log_entry()

raise_enter_downtime_log_entry()

raise_exit_downtime_log_entry()

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'modified_at

want_host_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

want_service_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

class shinken.objects.contact.**Contacts**(*items*)
Bases: *shinken.objects.item.Items*

explode(*contactgroups*, *notificationways*)

inner_class
alias of *Contact*

late_linkify_c_by_commands(*commands*)

linkify(*timeperiods*, *commands*, *notificationways*)

linkify_with_notificationways(*notificationways*)

name_property = 'contact_name'

contactgroup Module

class shinken.objects.contactgroup.**Contactgroup**(*params={}*)
Bases: *shinken.objects.itemgroup.Itemgroup*

get_contactgroup_members()

get_contacts()

get_contacts_by_explosion(*contactgroups*)

get_name()

id = 1

macros = {'CONTACTGROUPALIAS': 'alias', 'CONTACTGROUPMEMBERS': 'get_members'}

my_type = 'contactgroup'

properties = {'contactgroup_name': <shinken.property.StringProp object at 0x7fa5c40c1510>, 'use': <shinken.proper

class shinken.objects.contactgroup.**Contactgroups**(*items*)
Bases: *shinken.objects.itemgroup.Itemgroups*

add_contactgroup(*cg*)

add_member(*cname*, *cgname*)

explode()

get_members_by_name(*cgname*)

inner_class
alias of *Contactgroup*

linkify (*contacts*)

linkify_cg_by_cont (*contacts*)

name_property = 'contactgroup_name'

discoveryrule Module

class shinken.objects.discoveryrule.**Discoveryrule** (*params={}*)
Bases: *shinken.objects.matchingitem.MatchingItem*

get_name ()

id = 1

macros = {}

my_type = 'discoveryrule'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c4123950>}

class shinken.objects.discoveryrule.**Discoveryrules** (*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *Discoveryrule*

name_property = 'discoveryrule_name'

discoveryrun Module

class shinken.objects.discoveryrun.**Discoveryrun** (*params={}*)
Bases: *shinken.objects.matchingitem.MatchingItem*

check_finished ()

get_name ()

get_output ()

id = 1

is_finished ()

is_first_level ()

launch (*ctx=[], timeout=300*)

my_type = 'discoveryrun'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'discoveryrun_command': <shinken.prop

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.discoveryrun.**Discoveryruns** (*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *Discoveryrun*

linkify (*commands*)

name_property = 'discoveryrun_name'

escalation Module

class shinken.objects.escalation.**Escalation** (*params={}*)
Bases: *shinken.objects.item.Item*

get_name ()

get_next_notif_time (*t_wished*, *status*, *creation_time*, *interval*)

id = 1

is_correct ()

is_eligible (*t*, *status*, *notif_number*, *in_notif_time*, *interval*)

my_type = 'escalation'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.escalation.**Escalations** (*items*)
Bases: *shinken.objects.item.Items*

add_escalation (*es*)

explode (*hosts*, *hostgroups*, *contactgroups*)

inner_class
alias of *Escalation*

linkify (*timeperiods*, *contacts*, *services*, *hosts*)

linkify_es_by_h (*hosts*)

linkify_es_by_s (*services*)

name_property = 'escalation_name'

host Module

This is the main class for the Host. In fact it's mainly about the configuration part. for the running one, it's better to look at the schedulingitem class that manage all scheduling/consume check smart things :)

class shinken.objects.host.**Host** (*params={}*)
Bases: *shinken.objects.schedulingitem.SchedulingItem*

acknowledgement

acknowledgement_type

act_depend_of

act_depend_of_me

action_url

actions

active_checks_enabled

add_business_rule_act_dependency (*h*, *status*, *timeperiod*, *inherits_parent*)

add_host_act_dependency (*h*, *status*, *timeperiod*, *inherits_parent*)

add_host_chk_dependency (*h*, *status*, *timeperiod*, *inherits_parent*)

add_service_link (*service*)

address

alias

attempt

broks

business_impact

business_impact_modulations

business_rule

check_command

check_flapping_recovery_notification

check_freshness

check_interval

check_period

check_type

checkmodulations

checks_in_progress

child_dependencies

childs

chk_depend_of

chk_depend_of_me

comments

configuration_errors

configuration_warnings

contact_groups

contacts

current_event_id

current_notification_id

current_notification_number

current_problem_id

custom_views

customs

del_host_act_dependency (*other*)

display_name

downtimes

duration_sec

early_timeout

end_time

escalations

event_handler

event_handler_enabled

execution_time

failure_prediction_enabled

fill_parents_dependency()

fill_predictive_missing_parameters()

find_service_by_name(*service_description*)

first_notification_delay

flap_detection_enabled

flap_detection_options

flapping_changes

flapping_comment_id

freshness_threshold

get_ack_author_name()

get_ack_comment()

get_check_command()

get_data_for_checks()

get_data_for_event_handler()

get_data_for_notifications(*contact*, *n*)

get_dbg_name()

get_duration()

get_duration_sec()

get_full_name()

get_host_tags()

get_hostgroups()

get_name()

get_obsessive_compulsive_processor_command()

get_realm()

get_total_services()

get_total_services_critical()

get_total_services_ok()

get_total_services_unknown()

get_total_services_warning()

got_business_rule

got_default_realm

has_been_checked

hash

high_flap_threshold

host_name

hostgroups

icon_image

icon_image_alt

icon_set

id = 1

impacts

imported_from

in_checking

in_hard_unknown_reach_phase

in_scheduled_downtime

in_scheduled_downtime_during_last_check

initial_state

is_correct()

is_flapping

is_impact

is_linked_with_host(*other*)

is_problem

is_state(*status*)

last_chk

last_event_id

last_hard_state

last_hard_state_change

last_hard_state_id

last_notification

last_perf_data

last_problem_id

last_state

last_state_change

last_state_id

last_state_type

last_state_update

last_time_down

last_time_non_ok_or_up()

last_time_unreachable

last_time_up

latency

long_output

low_flap_threshold

macromodulations

macros = {'HOSTPERFDATA': 'perf_data', 'HOSTGROUPNAMES': 'get_groupnames', 'HOSTCHECKCOMMAND'

maintenance_period

manage_stalking(*c*)

max_check_attempts

modified_attributes

my_own_business_impact

my_type = 'host'

name

next_chk

notes

notes_url

notification_interval

notification_is_blocked_by_contact(*n*, *contact*)

notification_is_blocked_by_item(*type*, *t_wished=None*)

notification_options

notification_period

notifications_enabled

notifications_in_progress

notified_contacts

obsess_over_host

ok_up = 'UP'

old_properties = {'normal_check_interval': 'check_interval', 'hostgroup': 'hostgroups', 'criticity': 'business_impact

output

pack_id

parent_dependencies

parents

passive_checks_enabled

pending_flex_downtime

percent_state_change

perf_data

poller_tag

problem_has_been_acknowledged

process_perf_data

properties = {'statusmap_image': <shinken.property.StringProp object at 0x7fa5c3d58cd0>, 'icon_image_alt': <shink

raise_alert_log_entry()

raise_cancel_downtime_log_entry()

raise_enter_downtime_log_entry()

raise_event_handler_log_entry(*command*)

raise_exit_downtime_log_entry()

raise_flapping_start_log_entry(*change_ratio*, *threshold*)

raise_flapping_stop_log_entry(*change_ratio*, *threshold*)

raise_freshness_log_entry(*t_stale_by*, *t_threshold*)

raise_initial_state()

raise_no_next_check_log_entry()

raise_notification_log_entry(*n*)

reactionner_tag

realm

register_child(*child*)

resultmodulations

retain_nonstatus_information

retain_status_information

retry_interval

return_code

running_properties = {'state_id_before_impact': <shinken.property.StringProp object at 0x7fa5c3d4b550>, 'child_c

scheduled_downtime_depth

services

set_impact_state()

set_state_from_exit_status(*status*)

set_unreachable()

should_be_scheduled

source_problems

stalking_options

start_time

state

state_before_hard_unknown_reach_phase

state_before_impact

state_changed_since_impact

state_id

state_id_before_impact

state_type

state_type_id

statusmap_image

tags

time_to_orphanage

timeout

topology_change

trending_policies

trigger

trigger_name

triggers

unset_impact_state()

use

vrml_image

was_in_hard_unknown_reach_phase

class shinken.objects.host.**Hosts**(*items*)

Bases: *shinken.objects.item.Items*

apply_dependencies()

create_business_rules(*hosts*, *services*)

create_business_rules_dependencies()

explode(*hostgroups*, *contactgroups*, *triggers*)

fill_predictive_missing_parameters()

find_hosts_that_use_template(*tpl_name*)

inner_class

alias of *Host*

late_linkify_h_by_commands(*commands*)

linkify(*timeperiods=None*, *commands=None*, *contacts=None*, *realms=None*, *resultmodulations=None*, *businessimpactmodulations=None*, *escalations=None*, *hostgroups=None*, *triggers=None*, *checkmodulations=None*, *macromodulations=None*)

linkify_h_by_h()

linkify_h_by_hg(*hostgroups*)

linkify_h_by_realms(*realms*)

name_property = 'host_name'

no_loop_in_parents()

hostdependency Module

class shinken.objects.hostdependency.**Hostdependencies**(*items*)
Bases: *shinken.objects.item.Items*

apply_inheritance()

delete_hostsdep_by_id(*ids*)

explode(*hostgroups*)

linkify(*hosts*, *timeperiods*)

linkify_h_by_hd()

linkify_hd_by_h(*hosts*)

linkify_hd_by_tp(*timeperiods*)

class shinken.objects.hostdependency.**Hostdependency**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 0

my_type = 'hostdependency'

properties = {'inherits_parent': <shinken.property.BoolProp object at 0x7fa5c40c40d0>, 'use': <shinken.property.Lis

hostescalation Module

class shinken.objects.hostescalation.**Hostescalation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

my_type = 'hostescalation'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

class shinken.objects.hostescalation.**Hostescalations**(*items*)
Bases: *shinken.objects.item.Items*

explode(*escalations*)

inner_class
alias of *Hostescalation*

name_property = ''

hostextinfo Module

This is the main class for the Host ext info. In fact it's mainly about the configuration part. Parameters are merged in Hosts so it's no use in running part

class shinken.objects.hostextinfo.**HostExtInfo**(*params={}*)

Bases: *shinken.objects.item.Item*

get_dbg_name()

get_full_name()

get_name()

host_name

icon_image

icon_image_alt

id = 1

imported_from

is_correct()

macros = {'HOSTNOTES': 'notes', 'HOSTNAME': 'host_name', 'HOSTNOTESURL': 'notes_url'}

my_type = 'hostextinfo'

name

notes

notes_url

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

statusmap_image

use

vrml_image

class shinken.objects.hostextinfo.**HostsExtInfo**(*items*)

Bases: *shinken.objects.item.Items*

inner_class

alias of *HostExtInfo*

merge(*hosts*)

merge_extinfo(*host*, *extinfo*)

name_property = 'host_name'

hostgroup Module

class shinken.objects.hostgroup.**Hostgroup**(*params={}*)

Bases: *shinken.objects.itemgroup.Itemgroup*

get_hostgroup_members()

get_hosts()

get_hosts_by_explosion(*hostgroups*)

get_name()

id = 1

macros = {'HOSTGROUPMEMBERS': 'members', 'HOSTGROUPNOTES': 'notes', 'HOSTGROUPACTIONURL': 'a

my_type = 'hostgroup'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'hostgroup_name': <shinken.property.Str

class shinken.objects.hostgroup.**Hostgroups**(*items*)

Bases: *shinken.objects.itemgroup.Itemgroups*

add_member(*hname*, *hgname*)

explode()

get_members_by_name(*hgname*)

inner_class

alias of *Hostgroup*

linkify(*hosts=None*, *realms=None*)

linkify_hg_by_hst(*hosts*)

linkify_hg_by_realms(*realms*)

name_property = 'hostgroup_name'

item Module

This class is a base class for nearly all configuration elements like service, hosts or contacts.

class shinken.objects.item.**Item**(*params={}*)

Bases: object

acknowledge_problem(*sticky*, *notify*, *persistent*, *author*, *comment*, *end_time=0*)

add_comment(*comment*)

add_downtime(*downtime*)

check_for_expire_acknowledge()

clean()

Clean useless things not requested once item has been fully initialized&configured. Like temporary attributes such as "imported_from", etc..

compute_hash()

copy()

Return a copy of the item, but give him a new id

del_comment(*comment_id*)

del_downtime(*downtime_id*)

explode_trigger_string_into_triggers(*triggers*)

fill_data_brok_from(*data*, *brok_type*)

fill_default()

Fill missing properties if they are missing

get_all_plus_and_delete()

get_check_result_brok()

get_customs_properties_by_inheritance(*items*)

get_initial_status_brok()

get_next_schedule_brok()

get_plus_and_delete(*prop*)

get_property_by_inheritance(*items*, *prop*)

get_property_value_for_brok(*prop*, *tab*)

get_raw_import_values()

get_templates()

get_update_status_brok()

has_plus(*prop*)

init_running_properties()

is_correct()

is_tpl()
Return if the elements is a template

linkify_one_command_with_commands(*commands*, *prop*)

linkify_with_triggers(*triggers*)

classmethod load_global_conf(*conf*)
Used to put global values in the sub Class like hosts or services

macros = {}

old_properties_names_to_new()

prepare_for_conf_sending()

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

pythonize()

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'hash': <shin

unacknowledge_problem()

unacknowledge_problem_if_not_sticky()

class shinken.objects.item.**Items**(*items*)
Bases: object

apply_inheritance()

apply_partial_inheritance(*prop*)

clean()
Request to remove the unnecessary attributes/others from our items

compute_hash()

create_reversed_list()

create_tpl_list()

evaluate_hostgroup_expression(*expr*, *hosts*, *hostgroups*, *look_in='hostgroups'*)

explode_contact_groups_into_contacts(*contactgroups*)

explode_host_groups_into_hosts(*hosts*, *hostgroups*)

explode_trigger_string_into_triggers(*triggers*)

fill_default()

find_by_name(*name*)

find_id_by_name(*name*)

find_tpl_by_name(*name*)

is_correct()

linkify_command_list_with_commands(*commands*, *prop*)

linkify_one_command_with_commands(*commands*, *prop*)

linkify_templates()

linkify_with_business_impact_modulations(*business_impact_modulations*)

linkify_with_checkmodulations(*checkmodulations*)

linkify_with_contacts(*contacts*)

linkify_with_escalations(*escalations*)

linkify_with_macromodulations(*macromodulations*)

linkify_with_resultmodulations(*resultmodulations*)

linkify_with_timeperiods(*timeperiods*, *prop*)

linkify_with_triggers(*triggers*)

old_properties_names_to_new()

prepare_for_sending()

pythonize()

remove_templates()

Remove useless templates (& properties) of our items ; otherwise we could get errors on config.is_correct()

remove_twins()

`itemgroup` Module

class shinken.objects.itemgroup.**Itemgroup**(*params={}*)

Bases: *shinken.objects.item.Item*

add_string_member(*member*)

copy_shell()

fill_default()

get_initial_status_brok()

has(*prop*)

id = **0**

is_correct()

properties = {'members': <shinken.property.StringProp object at 0x7fa5c408a190>, 'unknown_members': <shinken.p

pythonize()

replace_members(*members*)

class shinken.objects.itemgroup.**Itemgroups**(*items*)
Bases: *shinken.objects.item.Items*

add(*ig*)

fill_default()

get_members_by_name(*gname*)

macromodulation Module

class shinken.objects.macromodulation.**MacroModulation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

is_active()

is_correct()

macros = {}

my_type = 'macromodulation'

properties = {'macromodulation_name': <shinken.property.StringProp object at 0x7fa5c436d350>, 'use': <shinken.pr

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.macromodulation.**MacroModulations**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *MacroModulation*

linkify(*timeperiods*)

name_property = 'macromodulation_name'

matchingitem Module

This is a utility class for factorizing matching functions for discovery runners and rules.

class shinken.objects.matchingitem.**MatchingItem**(*params={}*)
Bases: *shinken.objects.item.Item*

is_matching(*key*, *value*, *look_in='matches'*)

is_matching_disco_datas(*datas*)

module Module

class shinken.objects.module.**Module**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

macros = {}

my_type = 'module'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'module_type': <shinken.property.String

class shinken.objects.module.**Modules**(*items*)
Bases: *shinken.objects.item.Items*

explode()

inner_class
alias of *Module*

linkify()

linkify_s_by_plug()

name_property = 'module_name'

notificationway Module

class shinken.objects.notificationway.**NotificationWay**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

get_notification_commands(*type*)

id = 1

is_correct()

late_linkify_nw_by_commands(*commands*)

macros = {}

my_type = 'notificationway'

old_properties = {'min_criticity': 'min_business_impact'}

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'notificationway_name': <shinken.proper

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

want_host_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

want_service_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

class shinken.objects.notificationway.**NotificationWays**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *NotificationWay*

linkify(*timeperiods*, *commands*)

name_property = 'notificationway_name'

new_inner_member(*name=None*, *params={}*)

pack Module

class shinken.objects.pack.**Pack**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

my_type = 'pack'

properties = {'pack_name': <shinken.property.StringProp object at 0x7fa5c3da7e50>, 'use': <shinken.property.ListP

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.pack.**Packs**(*items*)
Bases: *shinken.objects.item.Items*

create_pack(*buf*, *name*)

inner_class
alias of *Pack*

load_file(*path*)

name_property = 'pack_name'

realm Module

class shinken.objects.realm.**Realm**(*params={}*)
Bases: *shinken.objects.itemgroup.Itemgroup*

add_string_member(*member*)

count_brokers()

count_pollers()

count_reactionners()

count_receivers()

fill_broker_with_poller_reactionner_links(*broker*)

fill_potential_brokers()

fill_potential_pollers()

fill_potential_reactionners()

fill_potential_receivers()

get_all_subs_pollers()

get_all_subs_reactionners()

get_name()

get_nb_of_must_have_satellites(*type*)

get_potential_satellites_by_type(*type*)

get_realm_members()

get_realms()

get_realms_by_explosion(*realms*)

get_satellites_links_for_scheduler()

get_satellties_by_type(*type*)

id = 1

macros = {'REALMNAME': 'realm_name', 'REALMMEMBERS': 'members'}

my_type = 'realm'

prepare_for_satellites_conf()

properties = {'higher_realms': <shinken.property.StringProp object at 0x7fa5c408ad90>, 'use': <shinken.property.Lis

pythonize()

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.realm.**Realms**(*items*)
Bases: *shinken.objects.itemgroup.Itemgroups*

explode()

get_default()

get_members_by_name(*pname*)

inner_class
alias of *Realm*

linkify()

linkify_p_by_p()

name_property = 'realm_name'

prepare_for_satellites_conf()

resultmodulation Module

class shinken.objects.resultmodulation.**Resultmodulation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

module_return(*return_code*)

my_type = 'resultmodulation'

properties = {'exit_code_modulation': <shinken.property.StringProp object at 0x7fa5c3d59850>, 'use': <shinken.prop

pythonize()

class shinken.objects.resultmodulation.**Resultmodulations**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *Resultmodulation*

linkify(*timeperiods*)

linkify_rm_by_tp(*timeperiods*)

name_property = 'resultmodulation_name'

schedulingitem Module

This class is a common one for service/host. Here you will find all scheduling related functions, like the schedule or the consume_check. It's a very important class!

class shinken.objects.schedulingitem.**SchedulingItem**(*params={}*)

Bases: *shinken.objects.item.Item*

add_attempt()

add_flapping_change(*b*)

check_and_set_unreachability()

check_for_flexible_downtime()

compensate_system_time_change(*difference*)

consume_result(*c*)

create_business_rules(*hosts*, *services*)

create_business_rules_dependencies()

create_notifications(*type*, *t_wished=None*)

current_event_id = **0**

current_problem_id = **0**

deregister_a_problem(*pb*)

disable_active_checks()

do_check_freshness()

do_i_raise_dependency(*status*, *inherit_parents*)

eval_triggers()

get_escalable_contacts(*n*)

get_event_handlers(*externalcmd=False*)

get_next_notification_time(*n*)

get_perfdata_command()

get_time_to_orphanage()

is_escalable(*n*)

is_max_attempts()

is_no_action_dependent()

is_no_check_dependent()

launch_check(*t*, *ref_check=None*, *force=False*)

manage_internal_check(*c*)

no_more_a_problem()

prepare_notification_for_sending(*n*)

raise_dependencies_check(*ref_check*)

rebuild_ref()

Rebuild the possible reference a schedulingitem can have

register_a_problem(*pb*)

register_son_in_parent_child_dependencies(*son*)

remove_in_progress_check(*c*)

remove_in_progress_notification(*n*)

remove_in_progress_notifications()

scatter_notification(*n*)

schedule(*force=False*, *force_time=None*)

set_myself_as_problem()

update_business_impact_value()

update_event_and_problem_id()

update_flapping()

update_hard_unknown_phase_state()

update_in_checking()

update_notification_command(*n*)

service Module

This Class is the service one, s it manage all service specific thing. If you look at the scheduling part, look at the scheduling item class

class shinken.objects.service.**Service**(*params={}*)

Bases: *shinken.objects.schedulingitem.SchedulingItem*

acknowledgement

acknowledgement_type

act_depend_of

act_depend_of_me

action_url

actions

active_checks_enabled

add_business_rule_act_dependency(*srv*, *status*, *timeperiod*, *inherits_parent*)

add_service_act_dependency(*srv*, *status*, *timeperiod*, *inherits_parent*)

add_service_chk_dependency(*srv*, *status*, *timeperiod*, *inherits_parent*)

aggregation

attempt

broks

business_impact

business_impact_modulations

business_rule

check_command

check_flapping_recovery_notification

check_freshness

check_interval

check_period

check_type

checkmodulations

checks_in_progress

child_dependencies

chk_depend_of

chk_depend_of_me

comments

configuration_errors

configuration_warnings

contact_groups

contacts

current_event_id

current_notification_id

current_notification_number

current_problem_id

custom_views

customs

default_value

display_name

downtimes

duplicate(*host*)

duplicate_foreach

duration_sec

early_timeout

end_time

escalations

event_handler

event_handler_enabled

execution_time

failure_prediction_enabled

fill_daddy_dependency()

first_notification_delay

flap_detection_enabled

flap_detection_options

flapping_changes

flapping_comment_id

freshness_threshold

get_ack_author_name()

get_ack_comment()

get_check_command()

get_data_for_checks()

get_data_for_event_handler()

get_data_for_notifications(*contact*, *n*)

get_dbg_name()

get_duration()

get_duration_sec()

get_full_name()

get_groupnames()

get_host_tags()

get_hostgroups()

get_name()

get_obsessive_compulsive_processor_command()

get_realm()

got_business_rule

has_been_checked

hash

high_flap_threshold

host

host_name

hostgroup_name

icon_image

icon_image_alt

icon_set

id = 1

impacts

imported_from

in_checking

in_hard_unknown_reach_phase

in_scheduled_downtime

in_scheduled_downtime_during_last_check

initial_state

is_correct()

is_flapping

is_impact

is_problem

is_state(*status*)

is_volatile

last_chk

last_event_id

last_hard_state

last_hard_state_change

last_hard_state_id

last_notification

last_perf_data

last_problem_id

last_state

last_state_change

last_state_id

last_state_type

last_state_update

last_time_critical

last_time_non_ok_or_up()

last_time_ok

last_time_unknown

last_time_warning

latency

long_output

low_flap_threshold

macromodulations

macros = {'LASTSERVICESTATEID': 'last_state_id', 'SERVICEDURATION': 'get_duration', 'SERVICEGROUPNAM

maintenance_period

manage_stalking(*c*)

max_check_attempts

modified_attributes

my_own_business_impact

my_type = 'service'

name

next_chk

notes

notes_url

notification_interval

notification_is_blocked_by_contact(*n*, *contact*)

notification_is_blocked_by_item(*type*, *t_wished=None*)

notification_options

notification_period

notifications_enabled

notifications_in_progress

notified_contacts

obsess_over_service

ok_up = 'OK'

old_properties = {'normal_check_interval': 'check_interval', 'hostgroups': 'hostgroup_name', 'hostgroup': 'hostgro

output

parallelize_check

parent_dependencies

passive_checks_enabled

pending_flex_downtime

percent_state_change

perf_data

poller_tag

problem_has_been_acknowledged

process_perf_data

properties = {'active_checks_enabled': <shinken.property.BoolProp object at 0x7fa5c3d7a650>, 'icon_image_alt': <sh

raise_alert_log_entry()

raise_cancel_downtime_log_entry()

raise_enter_downtime_log_entry()

raise_event_handler_log_entry(*command*)

raise_exit_downtime_log_entry()

raise_flapping_start_log_entry(*change_ratio*, *threshold*)

raise_flapping_stop_log_entry(*change_ratio*, *threshold*)

raise_freshness_log_entry(*t_stale_by*, *t_threshold*)

raise_initial_state()

raise_no_next_check_log_entry()

raise_notification_log_entry(*n*)

reactionner_tag

resultmodulations

retain_nonstatus_information

retain_status_information

retry_interval

return_code

running_properties = {'state_id_before_impact': <shinken.property.IntegerProp object at 0x7fa5c3d59e50>, 'child_

scheduled_downtime_depth

service_dependencies

service_description

servicegroups

set_impact_state()

set_state_from_exit_status(*status*)

set_unreachable()

should_be_scheduled

source_problems

stalking_options

start_time

state

state_before_hard_unknown_reach_phase

state_before_impact

state_changed_since_impact

state_id

state_id_before_impact

state_type

state_type_id

tags

time_to_orphanage

timeout

topology_change

trending_policies

trigger

trigger_name

triggers

unset_impact_state()

use

was_in_hard_unknown_reach_phase

class shinken.objects.service.**Services**(*items*)
Bases: *shinken.objects.item.Items*

apply_dependencies()

apply_implicit_inheritance(*hosts*)

apply_inheritance(*hosts*)

copy_create_service_from_another(*hosts*, *s*, *hname*)

create_business_rules(*hosts*, *services*)

create_business_rules_dependencies()

create_reversed_list()

delete_services_by_id(*ids*)

explode(*hosts*, *hostgroups*, *contactgroups*, *servicegroups*, *servicedependencies*, *triggers*)

find_srv_by_name_and_hostname(*host_name*, *name*)

find_srv_id_by_name_and_hostname(*host_name*, *name*)

inner_class
alias of *Service*

late_linkify_s_by_commands(*commands*)

linkify(*hosts*, *commands*, *timeperiods*, *contacts*, *resultmodulations*, *businessimpactmodulations*, *escalations*, *servicegroups*, *triggers*, *checkmodulations*, *macromodulations*)

linkify_s_by_hst(*hosts*)

linkify_s_by_sg(*servicegroups*)

optimize_service_search(*hosts*)

servicedependency Module

class shinken.objects.servicedependency.**Servicedependencies**(*items*)
Bases: *shinken.objects.item.Items*

add_service_dependency(*dep_host_name*, *dep_service_description*, *par_host_name*, *par_service_description*)

apply_inheritance(*hosts*)

delete_servicesdep_by_id(*ids*)

explode(*hostgroups*)

explode_hostgroup(*sd*, *hostgroups*)

linkify(*hosts*, *services*, *timeperiods*)

linkify_s_by_sd()

linkify_sd_by_s (*hosts*, *services*)

linkify_sd_by_tp (*timeperiods*)

class shinken.objects.servicedependency.**Servicedependency** (*params={}*)
Bases: *shinken.objects.item.Item*

get_name ()

id = 0

my_type = 'servicedependency'

properties = {'inherits_parent': <shinken.property.BoolProp object at 0x7fa5c40c4490>, 'use': <shinken.property.List

serviceescalation Module

class shinken.objects.serviceescalation.**Serviceescalation** (*params={}*)
Bases: *shinken.objects.item.Item*

get_name ()

id = 1

my_type = 'serviceescalation'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

class shinken.objects.serviceescalation.**Serviceescalations** (*items*)
Bases: *shinken.objects.item.Items*

explode (*escalations*)

inner_class
alias of *Serviceescalation*

name_property = ''

serviceextinfo Module

This is the main class for the Service ext info. In fact it's mainly about the configuration part. Parameters are merged in Service so it's no use in running part

class shinken.objects.serviceextinfo.**ServiceExtInfo** (*params={}*)
Bases: *shinken.objects.item.Item*

get_dbg_name ()

get_full_name ()

get_name ()

host_name

icon_image

icon_image_alt

id = 1

imported_from

is_correct ()

macros = {'SERVICENOTES': 'notes', 'SERVICEDESC': 'service_description', 'SERVICENOTESURL': 'notes_url', '

my_type = 'serviceextinfo'

name

notes

notes_url

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

service_description

use

class shinken.objects.serviceextinfo.**ServicesExtInfo**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *ServiceExtInfo*

merge(*services*)

merge_extinfo(*service*, *extinfo*)

name_property = 'host_name'

servicegroup Module

class shinken.objects.servicegroup.**Servicegroup**(*params={}*)
Bases: *shinken.objects.itemgroup.Itemgroup*

get_name()

get_servicegroup_members()

get_services()

get_services_by_explosion(*servicegroups*)

id = 1

macros = {'SERVICEGROUPNOTES': 'notes', 'SERVICEGROUPALIAS': 'alias', 'SERVICEGROUPMEMBERS': 'm

my_type = 'servicegroup'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

class shinken.objects.servicegroup.**Servicegroups**(*items*)
Bases: *shinken.objects.itemgroup.Itemgroups*

add_member(*cname*, *sgname*)

explode()

inner_class
alias of *Servicegroup*

linkify(*services*)

linkify_sg_by_srv(*services*)

name_property = 'servicegroup_name'

timeperiod Module

class shinken.objects.timeperiod.**Timeperiod**(*params={}*)

Bases: *shinken.objects.item.Item*

apply_inheritance()

check_and_log_activation_change()

check_exclude_rec()

clean_cache()

explode(*timeperiods*)

fill_data_brok_from(*data*, *brok_type*)

find_next_invalid_time_from_cache(*t*)

find_next_valid_time_from_cache(*t*)

get_initial_status_brok()

get_min_from_t(*t*)

get_name()

get_next_invalid_time_from_t(*t*)

get_next_valid_time_from_t(*t*)

get_not_in_min_from_t(*f*)

get_raw_import_values()

get_unresolved_properties_by_inheritance(*items*)

has(*prop*)

id = **1**

is_correct()

is_time_valid(*t*)

linkify(*timeperiods*)

my_type = **'timeperiod'**

properties = **{'use': <shinken.property.StringProp object at 0x7fa5c3ee05d0>, 'name': <shinken.property.StringProp**

resolve_daterange(*dateranges*, *entry*)

running_properties = **{'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio**

class shinken.objects.timeperiod.**Timeperiods**(*items*)

Bases: *shinken.objects.item.Items*

apply_inheritance()

explode()

inner_class

alias of *Timeperiod*

is_correct()

linkify()

name_property = **'timeperiod_name'**

trigger Module

class shinken.objects.trigger.**Trigger**(*params={}*)
Bases: *shinken.objects.item.Item*

compile()

eval(*myself*, *ctx*)

get_name()

id = 1

my_type = 'trigger'

properties = {'trigger_name': <shinken.property.StringProp object at 0x7fa5c5d36510>, 'use': <shinken.property.List

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.trigger.**Triggers**(*items*)
Bases: *shinken.objects.item.Items*

compile()

create_trigger(*src*, *name*)

inner_class
alias of *Trigger*

load_file(*path*)

load_objects(*conf*)

name_property = 'trigger_name'

webui Package

webui Package

bottlecore Module

bottlewebui Module

17.3 clients Package

17.3.1 clients Package

17.3.2 LSB Module

class shinken.clients.LSB.**FileQueryGenerator**(*filename*)
Bases: *shinken.clients.LSB.SimpleQueryGenerator*

class shinken.clients.LSB.**QueryGenerator**
Bases: object

Generate a livestatus query

get()

class shinken.clients.LSB.**SimpleQueryGenerator**(*querys*, *name='sqg'*)
Bases: *shinken.clients.LSB.QueryGenerator*

get()

shinken.clients.LSB.**main**(*argv*)

shinken.clients.LSB.**mean**(*numberList*)

shinken.clients.LSB.**median**(*numberList*)

shinken.clients.LSB.**run**(*url*, *requests*, *concurrency*, *qg*)

shinken.clients.LSB.**usage**()

17.3.3 livestatus Module

class shinken.clients.livestatus.**LSAsynConnection**(*addr='127.0.0.1'*, *port=50000*, *path=None*, *timeout=10*)
Bases: asyncore.dispatcher

do_connect()

do_read(*size*)

exec_command(*command*)

get_query()

get_returns()

handle_close()

handle_connect()

handle_read()

handle_write()

is_finished()

launch_raw_query(*query*)

look_for_timeout()

readable()

stack_query(*q*)

wait_returns()

writable()

class shinken.clients.livestatus.**LSConnectionPool**(*con_addrs*)
Bases: object

launch_raw_query(*query*)

class shinken.clients.livestatus.**LSSyncConnection**(*addr='127.0.0.1'*, *port=50000*, *path=None*, *timeout=10*)

connect()

exec_command(*command*)

launch_query(*query*)

read(*size*)

class shinken.clients.livestatus.**Query**(*q*)

Bases: object

get()

id = **0**

put(*r*)

17.4 daemons Package

17.4.1 daemons Package

17.4.2 arbiterdaemon Module

17.4.3 brokerdaemon Module

17.4.4 pollerdaemon Module

17.4.5 reactionnerdaemon Module

17.4.6 receiverdaemon Module

17.4.7 schedulerdaemon Module

17.5 discovery Package

17.5.1 discovery Package

17.5.2 discoverymanager Module

17.6 misc Package

17.6.1 misc Package

17.6.2 datamanager Module

class shinken.misc.datamanager.**DataManager**

Bases: object

get_all_hosts_and_services()

get_all_impacts()

get_all_problems(*to_sort=True*, *get_acknowledged=False*)

get_brokers()

get_business_parents(*obj*, *levels=3*)

get_contact(*name*)

get_contacts()

get_host(*hname*)

get_host_tags_sorted()

get_hostgroup(*name*)

get_hostgroups()

get_hostgroups_sorted(*selected=''*)

get_hosts()

get_important_elements()

get_important_impacts()

get_len_overall_state()

get_nb_all_problems()

get_nb_elements()

get_nb_impacts()

get_nb_problems()

get_overall_it_state()

get_overall_state()

get_per_hosts_state()

get_per_service_state()

get_pollers()

get_problems_time_sorted()

get_program_start()

get_reactionners()

get_realm(*r*)

get_realms()

get_receivers()

get_schedulers()

get_service(*hname*, *sdesc*)

get_services()

guess_root_problems(*obj*)

load(*rg*)

17.6.3 `filter` Module

Helper functions for some filtering, like for user based

`shinken.misc.filter.`**only_related_to**(*lst*, *user*)

17.6.4 `md5crypt` Module

md5crypt.py - Provides interoperable MD5-based crypt() function

SYNOPSIS

import md5crypt.py

cryptedpassword = md5crypt.md5crypt(password, salt);

DESCRIPTION

unix_md5_crypt() provides a crypt()-compatible interface to the rather new MD5-based crypt() function found in modern operating systems. It's based on the implementation found on FreeBSD 2.2.[56]-RELEASE and contains the following license in it:

> "THE BEER-WARE LICENSE" (Revision 42): <phk@login.dknet.dk> wrote this file. As long as you retain this notice you can do whatever you want with this stuff. If we meet some day, and you think this stuff is worth it, you can buy me a beer in return. Poul-Henning Kamp

apache_md5_crypt() provides a function compatible with Apache's .htpasswd files. This was contributed by Bryan Hart <bryan@eai.com>.

`shinken.misc.md5crypt.`**`apache_md5_crypt`**(*pw*, *salt*)

`shinken.misc.md5crypt.`**`md5crypt`**(*pw*, *salt*, *magic=None*)

`shinken.misc.md5crypt.`**`to64`**(*v*, *n*)

`shinken.misc.md5crypt.`**`unix_md5_crypt`**(*pw*, *salt*, *magic=None*)

17.6.5 `perfdata` Module

class `shinken.misc.perfdata.`**`Metric`**(*s*)

class `shinken.misc.perfdata.`**`PerfDatas`**(*s*)

`shinken.misc.perfdata.`**`guess_int_or_float`**(*val*)

17.6.6 `regenerator` Module

17.6.7 `sorter` Module

Helper functions for some sorting

`shinken.misc.sorter.`**`hst_srv_sort`**(*s1*, *s2*)

`shinken.misc.sorter.`**`last_state_change_earlier`**(*s1*, *s2*)

`shinken.misc.sorter.`**`worse_first`**(*s1*, *s2*)

17.6.8 `termcolor` Module

ANSII Color formatting for output in terminal.

`shinken.misc.termcolor.`**`colored`**(*text*, *color=None*, *on_color=None*, *attrs=None*)

Colorize text.

Available text colors: red, green, yellow, blue, magenta, cyan, white.

Available text highlights: on_red, on_green, on_yellow, on_blue, on_magenta, on_cyan, on_white.

Available attributes: bold, dark, underline, blink, reverse, concealed.

Example: colored('Hello, World!', 'red', 'on_grey', ['blue', 'blink']) colored('Hello, World!', 'green')

`shinken.misc.termcolor.`**`cprint`**(*text*, *color=None*, *on_color=None*, *attrs=None*, ***kwargs*)

Print colorize text.

It accepts arguments of print function.

17.7 objects Package

17.7.1 `objects` Package

The objects package contains definition classes of the different objects that can be declared in configuration files.

17.7.2 `businessimpactmodulation` Module

class `shinken.objects.businessimpactmodulation.`**`Businessimpactmodulation`**(*params={}*)

Bases: *`shinken.objects.item.Item`*

`get_name`()

`id` = **1**

`my_type` = **'businessimpactmodulation'**

`properties` = **{'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob**

class `shinken.objects.businessimpactmodulation.`**`Businessimpactmodulations`**(*items*)

Bases: *`shinken.objects.item.Items`*

`inner_class`

alias of *`Businessimpactmodulation`*

`linkify`(*timeperiods*)

`linkify_cm_by_tp`(*timeperiods*)

`name_property` = **'business_impact_modulation_name'**

17.7.3 `checkmodulation` Module

class `shinken.objects.checkmodulation.`**`CheckModulation`**(*params={}*)

Bases: *`shinken.objects.item.Item`*

`get_check_command`(*t_to_go*)

`get_name`()

`id` = **1**

`is_correct`()

`late_linkify_cw_by_commands`(*commands*)

`macros` = **{}**

`my_type` = **'checkmodulation'**

`properties` = **{'check_period': <shinken.property.StringProp object at 0x7fa5c436d490>, 'use': <shinken.property.List**

`running_properties` = **{'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio**

class shinken.objects.checkmodulation.**CheckModulations**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *CheckModulation*

linkify(*timeperiods*, *commands*)

name_property = 'checkmodulation_name'

new_inner_member(*name=None*, *params={}*)

17.7.4 command Module

class shinken.objects.command.**Command**(*params={}*)
Bases: *shinken.objects.item.Item*

command_line

command_name

fill_data_brok_from(*data*, *brok_type*)

get_initial_status_brok()

get_name()

id = 0

imported_from

module_type

my_type = 'command'

name

poller_tag

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

pythonize()

reactionner_tag

timeout

use

class shinken.objects.command.**Commands**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *Command*

name_property = 'command_name'

class shinken.objects.command.**DummyCommand**
Bases: object

17.7.5 config Module

17.7.6 contact Module

class shinken.objects.contact.**Contact**(*params={}*)

Bases: *shinken.objects.item.Item*

get_name()

get_notification_commands(*type*)

id = **1**

is_correct()

macros = **{'CONTACTPAGER': 'pager', 'CONTACTGROUPNAMES': 'get_groupnames', 'CONTACTGROUPNAME'**

my_type = **'contact'**

old_properties = **{'min_criticity': 'min_business_impact'}**

properties = **{'address1': <shinken.property.StringProp object at 0x7fa5c40b33d0>, 'address2': <shinken.property.Str**

raise_cancel_downtime_log_entry()

raise_enter_downtime_log_entry()

raise_exit_downtime_log_entry()

running_properties = **{'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'modified_att**

want_host_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

want_service_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

class shinken.objects.contact.**Contacts**(*items*)

Bases: *shinken.objects.item.Items*

explode(*contactgroups*, *notificationways*)

inner_class

alias of *Contact*

late_linkify_c_by_commands(*commands*)

linkify(*timeperiods*, *commands*, *notificationways*)

linkify_with_notificationways(*notificationways*)

name_property = **'contact_name'**

17.7.7 contactgroup Module

class shinken.objects.contactgroup.**Contactgroup**(*params={}*)

Bases: *shinken.objects.itemgroup.Itemgroup*

get_contactgroup_members()

get_contacts()

get_contacts_by_explosion(*contactgroups*)

get_name()

id = **1**

macros = {'CONTACTGROUPALIAS': 'alias', 'CONTACTGROUPMEMBERS': 'get_members'}

my_type = 'contactgroup'

properties = {'contactgroup_name': <shinken.property.StringProp object at 0x7fa5c40c1510>, 'use': <shinken.proper

class shinken.objects.contactgroup.**Contactgroups**(*items*)
Bases: *shinken.objects.itemgroup.Itemgroups*

add_contactgroup(*cg*)

add_member(*cname*, *cgname*)

explode()

get_members_by_name(*cgname*)

inner_class
alias of *Contactgroup*

linkify(*contacts*)

linkify_cg_by_cont(*contacts*)

name_property = 'contactgroup_name'

17.7.8 discoveryrule Module

class shinken.objects.discoveryrule.**Discoveryrule**(*params={}*)
Bases: *shinken.objects.matchingitem.MatchingItem*

get_name()

id = 1

macros = {}

my_type = 'discoveryrule'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c4123950>}

class shinken.objects.discoveryrule.**Discoveryrules**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *Discoveryrule*

name_property = 'discoveryrule_name'

17.7.9 discoveryrun Module

class shinken.objects.discoveryrun.**Discoveryrun**(*params={}*)
Bases: *shinken.objects.matchingitem.MatchingItem*

check_finished()

get_name()

get_output()

id = 1

is_finished()

is_first_level()

launch(*ctx=[]*, *timeout=300*)

my_type = 'discoveryrun'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'discoveryrun_command': <shinken.prop

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.discoveryrun.**Discoveryruns**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *Discoveryrun*

linkify(*commands*)

name_property = 'discoveryrun_name'

17.7.10 escalation Module

class shinken.objects.escalation.**Escalation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

get_next_notif_time(*t_wished*, *status*, *creation_time*, *interval*)

id = 1

is_correct()

is_eligible(*t*, *status*, *notif_number*, *in_notif_time*, *interval*)

my_type = 'escalation'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.escalation.**Escalations**(*items*)
Bases: *shinken.objects.item.Items*

add_escalation(*es*)

explode(*hosts*, *hostgroups*, *contactgroups*)

inner_class
alias of *Escalation*

linkify(*timeperiods*, *contacts*, *services*, *hosts*)

linkify_es_by_h(*hosts*)

linkify_es_by_s(*services*)

name_property = 'escalation_name'

17.7.11 host Module

This is the main class for the Host. In fact it's mainly about the configuration part. for the running one, it's better to look at the schedulingitem class that manage all scheduling/consume check smart things :)

class shinken.objects.host.**Host**(*params={}*)

Bases: *shinken.objects.schedulingitem.SchedulingItem*

acknowledgement

acknowledgement_type

act_depend_of

act_depend_of_me

action_url

actions

active_checks_enabled

add_business_rule_act_dependency(*h*, *status*, *timeperiod*, *inherits_parent*)

add_host_act_dependency(*h*, *status*, *timeperiod*, *inherits_parent*)

add_host_chk_dependency(*h*, *status*, *timeperiod*, *inherits_parent*)

add_service_link(*service*)

address

alias

attempt

broks

business_impact

business_impact_modulations

business_rule

check_command

check_flapping_recovery_notification

check_freshness

check_interval

check_period

check_type

checkmodulations

checks_in_progress

child_dependencies

childs

chk_depend_of

chk_depend_of_me

comments

configuration_errors

configuration_warnings

contact_groups

contacts

current_event_id

current_notification_id

current_notification_number

current_problem_id

custom_views

customs

del_host_act_dependency(*other*)

display_name

downtimes

duration_sec

early_timeout

end_time

escalations

event_handler

event_handler_enabled

execution_time

failure_prediction_enabled

fill_parents_dependency()

fill_predictive_missing_parameters()

find_service_by_name(*service_description*)

first_notification_delay

flap_detection_enabled

flap_detection_options

flapping_changes

flapping_comment_id

freshness_threshold

get_ack_author_name()

get_ack_comment()

get_check_command()

get_data_for_checks()

get_data_for_event_handler()

get_data_for_notifications(*contact*, *n*)

get_dbg_name()

get_duration()

get_duration_sec()

get_full_name()

get_host_tags()

get_hostgroups()

get_name()

get_obsessive_compulsive_processor_command()

get_realm()

get_total_services()

get_total_services_critical()

get_total_services_ok()

get_total_services_unknown()

get_total_services_warning()

got_business_rule

got_default_realm

has_been_checked

hash

high_flap_threshold

host_name

hostgroups

icon_image

icon_image_alt

icon_set

id = 1

impacts

imported_from

in_checking

in_hard_unknown_reach_phase

in_scheduled_downtime

in_scheduled_downtime_during_last_check

initial_state

is_correct()

is_flapping

is_impact

is_linked_with_host(*other*)

is_problem

is_state(*status*)

last_chk

last_event_id

last_hard_state

last_hard_state_change

last_hard_state_id

last_notification

last_perf_data

last_problem_id

last_state

last_state_change

last_state_id

last_state_type

last_state_update

last_time_down

last_time_non_ok_or_up()

last_time_unreachable

last_time_up

latency

long_output

low_flap_threshold

macromodulations

macros = {'HOSTPERFDATA': 'perf_data', 'HOSTGROUPNAMES': 'get_groupnames', 'HOSTCHECKCOMMAND'

maintenance_period

manage_stalking(*c*)

max_check_attempts

modified_attributes

my_own_business_impact

my_type = 'host'

name

next_chk

notes

notes_url

notification_interval

notification_is_blocked_by_contact(*n*, *contact*)

notification_is_blocked_by_item(*type*, *t_wished=None*)

notification_options

notification_period

notifications_enabled

notifications_in_progress

notified_contacts

obsess_over_host

ok_up = 'UP'

old_properties = {'normal_check_interval': 'check_interval', 'hostgroup': 'hostgroups', 'criticity': 'business_impact

output

pack_id

parent_dependencies

parents

passive_checks_enabled

pending_flex_downtime

percent_state_change

perf_data

poller_tag

problem_has_been_acknowledged

process_perf_data

properties = {'statusmap_image': <shinken.property.StringProp object at 0x7fa5c3d58cd0>, 'icon_image_alt': <shink

raise_alert_log_entry()

raise_cancel_downtime_log_entry()

raise_enter_downtime_log_entry()

raise_event_handler_log_entry(*command*)

raise_exit_downtime_log_entry()

raise_flapping_start_log_entry(*change_ratio*, *threshold*)

raise_flapping_stop_log_entry(*change_ratio*, *threshold*)

raise_freshness_log_entry(*t_stale_by*, *t_threshold*)

raise_initial_state()

raise_no_next_check_log_entry()

raise_notification_log_entry(*n*)

reactionner_tag

realm

register_child(*child*)

resultmodulations

retain_nonstatus_information

retain_status_information

retry_interval

return_code

running_properties = {'state_id_before_impact': <shinken.property.StringProp object at 0x7fa5c3d4b550>, 'child_c

scheduled_downtime_depth

services

set_impact_state()

set_state_from_exit_status(*status*)

set_unreachable()

should_be_scheduled

source_problems

stalking_options

start_time

state

state_before_hard_unknown_reach_phase

state_before_impact

state_changed_since_impact

state_id

state_id_before_impact

state_type

state_type_id

statusmap_image

tags

time_to_orphanage

timeout

topology_change

trending_policies

trigger

trigger_name

triggers

unset_impact_state()

use

vrml_image

was_in_hard_unknown_reach_phase

class shinken.objects.host.**Hosts**(*items*)

Bases: *shinken.objects.item.Items*

apply_dependencies()

create_business_rules(*hosts*, *services*)

create_business_rules_dependencies()

explode(*hostgroups*, *contactgroups*, *triggers*)

fill_predictive_missing_parameters()

find_hosts_that_use_template(*tpl_name*)

inner_class
alias of *Host*

late_linkify_h_by_commands(*commands*)

linkify(*timeperiods=None*, *commands=None*, *contacts=None*, *realms=None*, *resultmodulations=None*, *businessimpactmodulations=None*, *escalations=None*, *hostgroups=None*, *triggers=None*, *checkmodulations=None*, *macromodulations=None*)

linkify_h_by_h()

linkify_h_by_hg(*hostgroups*)

linkify_h_by_realms(*realms*)

name_property = 'host_name'

no_loop_in_parents()

17.7.12 hostdependency Module

class shinken.objects.hostdependency.**Hostdependencies**(*items*)
Bases: *shinken.objects.item.Items*

apply_inheritance()

delete_hostsdep_by_id(*ids*)

explode(*hostgroups*)

linkify(*hosts*, *timeperiods*)

linkify_h_by_hd()

linkify_hd_by_h(*hosts*)

linkify_hd_by_tp(*timeperiods*)

class shinken.objects.hostdependency.**Hostdependency**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 0

my_type = 'hostdependency'

properties = {'inherits_parent': <shinken.property.BoolProp object at 0x7fa5c40c40d0>, 'use': <shinken.property.Lis

17.7.13 hostescalation Module

class shinken.objects.hostescalation.**Hostescalation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

my_type = 'hostescalation'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

class shinken.objects.hostescalation.**Hostescalations**(*items*)
Bases: *shinken.objects.item.Items*

explode(*escalations*)

inner_class
alias of *Hostescalation*

name_property = ''

17.7.14 hostextinfo Module

This is the main class for the Host ext info. In fact it's mainly about the configuration part. Parameters are merged in Hosts so it's no use in running part

class shinken.objects.hostextinfo.**HostExtInfo**(*params={}*)
Bases: *shinken.objects.item.Item*

get_dbg_name()

get_full_name()

get_name()

host_name

icon_image

icon_image_alt

id = 1

imported_from

is_correct()

macros = {'HOSTNOTES': 'notes', 'HOSTNAME': 'host_name', 'HOSTNOTESURL': 'notes_url'}

my_type = 'hostextinfo'

name

notes

notes_url

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

statusmap_image

use

vrml_image

class shinken.objects.hostextinfo.**HostsExtInfo**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *HostExtInfo*

merge(*hosts*)

merge_extinfo(*host*, *extinfo*)

name_property = 'host_name'

17.7.15 hostgroup Module

class shinken.objects.hostgroup.**Hostgroup**(*params={}*)
Bases: *shinken.objects.itemgroup.Itemgroup*

get_hostgroup_members()

get_hosts()

get_hosts_by_explosion(*hostgroups*)

get_name()

id = 1

macros = {'HOSTGROUPMEMBERS': 'members', 'HOSTGROUPNOTES': 'notes', 'HOSTGROUPACTIONURL': 'a

my_type = 'hostgroup'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'hostgroup_name': <shinken.property.Str

class shinken.objects.hostgroup.**Hostgroups**(*items*)
Bases: *shinken.objects.itemgroup.Itemgroups*

add_member(*hname*, *hgname*)

explode()

get_members_by_name(*hgname*)

inner_class
alias of *Hostgroup*

linkify(*hosts=None*, *realms=None*)

linkify_hg_by_hst(*hosts*)

linkify_hg_by_realms(*realms*)

name_property = 'hostgroup_name'

17.7.16 item Module

This class is a base class for nearly all configuration elements like service, hosts or contacts.

class shinken.objects.item.**Item**(*params={}*)
Bases: object

acknowledge_problem(*sticky*, *notify*, *persistent*, *author*, *comment*, *end_time=0*)

add_comment(*comment*)

add_downtime(*downtime*)

check_for_expire_acknowledge()

clean()
Clean useless things not requested once item has been fully initialized&configured. Like temporary attributes such as "imported_from", etc..

compute_hash()

copy()
Return a copy of the item, but give him a new id

del_comment(*comment_id*)

del_downtime(*downtime_id*)

explode_trigger_string_into_triggers(*triggers*)

fill_data_brok_from(*data*, *brok_type*)

fill_default()
Fill missing properties if they are missing

get_all_plus_and_delete()

get_check_result_brok()

get_customs_properties_by_inheritance(*items*)

get_initial_status_brok()

get_next_schedule_brok()

get_plus_and_delete(*prop*)

get_property_by_inheritance(*items*, *prop*)

get_property_value_for_brok(*prop*, *tab*)

get_raw_import_values()

get_templates()

get_update_status_brok()

has_plus(*prop*)

init_running_properties()

is_correct()

is_tpl()
Return if the elements is a template

linkify_one_command_with_commands(*commands*, *prop*)

linkify_with_triggers(*triggers*)

classmethod load_global_conf(*conf*)
Used to put global values in the sub Class like hosts or services

macros = {}

old_properties_names_to_new()

prepare_for_conf_sending()

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

pythonize()

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'hash': <shin

unacknowledge_problem()

unacknowledge_problem_if_not_sticky()

class shinken.objects.item.**Items**(*items*)
Bases: object

apply_inheritance()

apply_partial_inheritance(*prop*)

clean()

Request to remove the unnecessary attributes/others from our items

compute_hash()

create_reversed_list()

create_tpl_list()

evaluate_hostgroup_expression(*expr*, *hosts*, *hostgroups*, *look_in='hostgroups'*)

explode_contact_groups_into_contacts(*contactgroups*)

explode_host_groups_into_hosts(*hosts*, *hostgroups*)

explode_trigger_string_into_triggers(*triggers*)

fill_default()

find_by_name(*name*)

find_id_by_name(*name*)

find_tpl_by_name(*name*)

is_correct()

linkify_command_list_with_commands(*commands*, *prop*)

linkify_one_command_with_commands(*commands*, *prop*)

linkify_templates()

linkify_with_business_impact_modulations(*business_impact_modulations*)

linkify_with_checkmodulations(*checkmodulations*)

linkify_with_contacts(*contacts*)

linkify_with_escalations(*escalations*)

linkify_with_macromodulations(*macromodulations*)

linkify_with_resultmodulations(*resultmodulations*)

linkify_with_timeperiods(*timeperiods*, *prop*)

linkify_with_triggers(*triggers*)

old_properties_names_to_new()

prepare_for_sending()

pythonize()

remove_templates()

Remove useless templates (& properties) of our items ; otherwise we could get errors on config.is_correct()

remove_twins()

17.7.17 itemgroup Module

class shinken.objects.itemgroup.**Itemgroup**(*params={}*)
Bases: *shinken.objects.item.Item*

add_string_member(*member*)

copy_shell()

fill_default()

get_initial_status_brok()

has(*prop*)

id = **0**

is_correct()

properties = **{'members': <shinken.property.StringProp object at 0x7fa5c408a190>, 'unknown_members': <shinken.p**

pythonize()

replace_members(*members*)

class shinken.objects.itemgroup.**Itemgroups**(*items*)
Bases: *shinken.objects.item.Items*

add(*ig*)

fill_default()

get_members_by_name(*gname*)

17.7.18 macromodulation Module

class shinken.objects.macromodulation.**MacroModulation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = **1**

is_active()

is_correct()

macros = **{}**

my_type = **'macromodulation'**

properties = **{'macromodulation_name': <shinken.property.StringProp object at 0x7fa5c436d350>, 'use': <shinken.pr**

running_properties = **{'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio**

class shinken.objects.macromodulation.**MacroModulations**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *MacroModulation*

linkify(*timeperiods*)

name_property = **'macromodulation_name'**

17.7.19 matchingitem Module

This is a utility class for factorizing matching functions for discovery runners and rules.

class shinken.objects.matchingitem.**MatchingItem**(*params={}*)

Bases: *shinken.objects.item.Item*

is_matching(*key*, *value*, *look_in='matches'*)

is_matching_disco_datas(*datas*)

17.7.20 module Module

class shinken.objects.module.**Module**(*params={}*)

Bases: *shinken.objects.item.Item*

get_name()

id = **1**

macros = **{}**

my_type = **'module'**

properties = **{'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'module_type': <shinken.property.String**

class shinken.objects.module.**Modules**(*items*)

Bases: *shinken.objects.item.Items*

explode()

inner_class

alias of *Module*

linkify()

linkify_s_by_plug()

name_property = **'module_name'**

17.7.21 notificationway Module

class shinken.objects.notificationway.**NotificationWay**(*params={}*)

Bases: *shinken.objects.item.Item*

get_name()

get_notification_commands(*type*)

id = **1**

is_correct()

late_linkify_nw_by_commands(*commands*)

macros = **{}**

my_type = **'notificationway'**

old_properties = **{'min_criticity': 'min_business_impact'}**

properties = **{'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'notificationway_name': <shinken.proper**

running_properties = **{'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio**

want_host_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

want_service_notification(*t*, *state*, *type*, *business_impact*, *cmd=None*)

class shinken.objects.notificationway.**NotificationWays**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *NotificationWay*

linkify(*timeperiods*, *commands*)

name_property = 'notificationway_name'

new_inner_member(*name=None*, *params={}*)

17.7.22 pack Module

class shinken.objects.pack.**Pack**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

my_type = 'pack'

properties = {'pack_name': <shinken.property.StringProp object at 0x7fa5c3da7e50>, 'use': <shinken.property.ListP

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.pack.**Packs**(*items*)
Bases: *shinken.objects.item.Items*

create_pack(*buf*, *name*)

inner_class
alias of *Pack*

load_file(*path*)

name_property = 'pack_name'

17.7.23 realm Module

class shinken.objects.realm.**Realm**(*params={}*)
Bases: *shinken.objects.itemgroup.Itemgroup*

add_string_member(*member*)

count_brokers()

count_pollers()

count_reactionners()

count_receivers()

fill_broker_with_poller_reactionner_links(*broker*)

fill_potential_brokers()

fill_potential_pollers()

fill_potential_reactionners()

fill_potential_receivers()

get_all_subs_pollers()

get_all_subs_reactionners()

get_name()

get_nb_of_must_have_satellites(*type*)

get_potential_satellites_by_type(*type*)

get_realm_members()

get_realms()

get_realms_by_explosion(*realms*)

get_satellites_links_for_scheduler()

get_satellties_by_type(*type*)

id = 1

macros = {'REALMNAME': 'realm_name', 'REALMMEMBERS': 'members'}

my_type = 'realm'

prepare_for_satellites_conf()

properties = {'higher_realms': <shinken.property.StringProp object at 0x7fa5c408ad90>, 'use': <shinken.property.Lis

pythonize()

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.realm.**Realms**(*items*)

Bases: *shinken.objects.itemgroup.Itemgroups*

explode()

get_default()

get_members_by_name(*pname*)

inner_class

alias of *Realm*

linkify()

linkify_p_by_p()

name_property = 'realm_name'

prepare_for_satellites_conf()

17.7.24 resultmodulation Module

class shinken.objects.resultmodulation.**Resultmodulation**(*params={}*)

Bases: *shinken.objects.item.Item*

get_name()

id = 1

module_return(*return_code*)

my_type = 'resultmodulation'

properties = {'exit_code_modulation': <shinken.property.StringProp object at 0x7fa5c3d59850>, 'use': <shinken.prop

pythonize()

class shinken.objects.resultmodulation.**Resultmodulations**(*items*)
Bases: *shinken.objects.item.Items*

inner_class
alias of *Resultmodulation*

linkify(*timeperiods*)

linkify_rm_by_tp(*timeperiods*)

name_property = 'resultmodulation_name'

17.7.25 schedulingitem Module

This class is a common one for service/host. Here you will find all scheduling related functions, like the schedule or the consume_check. It's a very important class!

class shinken.objects.schedulingitem.**SchedulingItem**(*params={}*)
Bases: *shinken.objects.item.Item*

add_attempt()

add_flapping_change(*b*)

check_and_set_unreachability()

check_for_flexible_downtime()

compensate_system_time_change(*difference*)

consume_result(*c*)

create_business_rules(*hosts*, *services*)

create_business_rules_dependencies()

create_notifications(*type*, *t_wished=None*)

current_event_id = 0

current_problem_id = 0

deregister_a_problem(*pb*)

disable_active_checks()

do_check_freshness()

do_i_raise_dependency(*status*, *inherit_parents*)

eval_triggers()

get_escalable_contacts(*n*)

get_event_handlers(*externalcmd=False*)

get_next_notification_time(*n*)

get_perfdata_command()

get_time_to_orphanage()

is_escalable(*n*)

is_max_attempts()

is_no_action_dependent()

is_no_check_dependent()

launch_check(*t*, *ref_check=None*, *force=False*)

manage_internal_check(*c*)

no_more_a_problem()

prepare_notification_for_sending(*n*)

raise_dependencies_check(*ref_check*)

rebuild_ref()
Rebuild the possible reference a schedulingitem can have

register_a_problem(*pb*)

register_son_in_parent_child_dependencies(*son*)

remove_in_progress_check(*c*)

remove_in_progress_notification(*n*)

remove_in_progress_notifications()

scatter_notification(*n*)

schedule(*force=False*, *force_time=None*)

set_myself_as_problem()

update_business_impact_value()

update_event_and_problem_id()

update_flapping()

update_hard_unknown_phase_state()

update_in_checking()

update_notification_command(*n*)

17.7.26 service Module

This Class is the service one, s it manage all service specific thing. If you look at the scheduling part, look at the scheduling item class

class shinken.objects.service.**Service**(*params={}*)
Bases: *shinken.objects.schedulingitem.SchedulingItem*

acknowledgement

acknowledgement_type

act_depend_of

act_depend_of_me

action_url

actions

active_checks_enabled

add_business_rule_act_dependency(*srv*, *status*, *timeperiod*, *inherits_parent*)

add_service_act_dependency(*srv*, *status*, *timeperiod*, *inherits_parent*)

add_service_chk_dependency(*srv*, *status*, *timeperiod*, *inherits_parent*)

aggregation

attempt

broks

business_impact

business_impact_modulations

business_rule

check_command

check_flapping_recovery_notification

check_freshness

check_interval

check_period

check_type

checkmodulations

checks_in_progress

child_dependencies

chk_depend_of

chk_depend_of_me

comments

configuration_errors

configuration_warnings

contact_groups

contacts

current_event_id

current_notification_id

current_notification_number

current_problem_id

custom_views

customs

default_value

display_name

downtimes

duplicate(*host*)

duplicate_foreach

duration_sec

early_timeout

end_time

escalations

event_handler

event_handler_enabled

execution_time

failure_prediction_enabled

fill_daddy_dependency()

first_notification_delay

flap_detection_enabled

flap_detection_options

flapping_changes

flapping_comment_id

freshness_threshold

get_ack_author_name()

get_ack_comment()

get_check_command()

get_data_for_checks()

get_data_for_event_handler()

get_data_for_notifications(*contact*, *n*)

get_dbg_name()

get_duration()

get_duration_sec()

get_full_name()

get_groupnames()

get_host_tags()

get_hostgroups()

get_name()

get_obsessive_compulsive_processor_command()

get_realm()

got_business_rule

has_been_checked

hash

high_flap_threshold

host

host_name

hostgroup_name

icon_image

icon_image_alt

icon_set

id = 1

impacts

imported_from

in_checking

in_hard_unknown_reach_phase

in_scheduled_downtime

in_scheduled_downtime_during_last_check

initial_state

is_correct()

is_flapping

is_impact

is_problem

is_state(*status*)

is_volatile

last_chk

last_event_id

last_hard_state

last_hard_state_change

last_hard_state_id

last_notification

last_perf_data

last_problem_id

last_state

last_state_change

last_state_id

last_state_type

last_state_update

last_time_critical

last_time_non_ok_or_up()

last_time_ok

last_time_unknown

last_time_warning

latency

long_output

low_flap_threshold

macromodulations

macros = {'LASTSERVICESTATEID': 'last_state_id', 'SERVICEDURATION': 'get_duration', 'SERVICEGROUPNAM

maintenance_period

manage_stalking(*c*)

max_check_attempts

modified_attributes

my_own_business_impact

my_type = 'service'

name

next_chk

notes

notes_url

notification_interval

notification_is_blocked_by_contact(*n*, *contact*)

notification_is_blocked_by_item(*type*, *t_wished=None*)

notification_options

notification_period

notifications_enabled

notifications_in_progress

notified_contacts

obsess_over_service

ok_up = 'OK'

old_properties = {'normal_check_interval': 'check_interval', 'hostgroups': 'hostgroup_name', 'hostgroup': 'hostgro

output

parallelize_check

parent_dependencies

passive_checks_enabled

pending_flex_downtime

percent_state_change

perf_data

poller_tag

problem_has_been_acknowledged

process_perf_data

properties = {'active_checks_enabled': <shinken.property.BoolProp object at 0x7fa5c3d7a650>, 'icon_image_alt': <sh

raise_alert_log_entry()

raise_cancel_downtime_log_entry()

raise_enter_downtime_log_entry()

raise_event_handler_log_entry(*command*)

raise_exit_downtime_log_entry()

raise_flapping_start_log_entry(*change_ratio*, *threshold*)

raise_flapping_stop_log_entry(*change_ratio*, *threshold*)

raise_freshness_log_entry(*t_stale_by*, *t_threshold*)

raise_initial_state()

raise_no_next_check_log_entry()

raise_notification_log_entry(*n*)

reactionner_tag

resultmodulations

retain_nonstatus_information

retain_status_information

retry_interval

return_code

running_properties = {'state_id_before_impact': <shinken.property.IntegerProp object at 0x7fa5c3d59e50>, 'child_

scheduled_downtime_depth

service_dependencies

service_description

servicegroups

set_impact_state()

set_state_from_exit_status(*status*)

set_unreachable()

should_be_scheduled

source_problems

stalking_options

start_time

state

state_before_hard_unknown_reach_phase

state_before_impact

state_changed_since_impact

state_id

state_id_before_impact

state_type

state_type_id

tags

time_to_orphanage

timeout

topology_change

trending_policies

trigger

trigger_name

triggers

unset_impact_state()

use

was_in_hard_unknown_reach_phase

class shinken.objects.service.**Services**(*items*)

Bases: *shinken.objects.item.Items*

apply_dependencies()

apply_implicit_inheritance(*hosts*)

apply_inheritance(*hosts*)

copy_create_service_from_another(*hosts*, *s*, *hname*)

create_business_rules(*hosts*, *services*)

create_business_rules_dependencies()

create_reversed_list()

delete_services_by_id(*ids*)

explode(*hosts*, *hostgroups*, *contactgroups*, *servicegroups*, *servicedependencies*, *triggers*)

find_srv_by_name_and_hostname(*host_name*, *name*)

find_srv_id_by_name_and_hostname(*host_name*, *name*)

inner_class

alias of *Service*

late_linkify_s_by_commands(*commands*)

linkify(*hosts*, *commands*, *timeperiods*, *contacts*, *resultmodulations*, *businessimpactmodulations*, *escalations*, *servicegroups*, *triggers*, *checkmodulations*, *macromodulations*)

linkify_s_by_hst(*hosts*)

linkify_s_by_sg(*servicegroups*)

optimize_service_search(*hosts*)

17.7.27 servicedependency Module

class shinken.objects.servicedependency.**Servicedependencies**(*items*)
Bases: *shinken.objects.item.Items*

add_service_dependency(*dep_host_name*, *dep_service_description*, *par_host_name*, *par_service_description*)

apply_inheritance(*hosts*)

delete_servicesdep_by_id(*ids*)

explode(*hostgroups*)

explode_hostgroup(*sd*, *hostgroups*)

linkify(*hosts*, *services*, *timeperiods*)

linkify_s_by_sd()

linkify_sd_by_s(*hosts*, *services*)

linkify_sd_by_tp(*timeperiods*)

class shinken.objects.servicedependency.**Servicedependency**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 0

my_type = 'servicedependency'

properties = {'inherits_parent': <shinken.property.BoolProp object at 0x7fa5c40c4490>, 'use': <shinken.property.List

17.7.28 serviceescalation Module

class shinken.objects.serviceescalation.**Serviceescalation**(*params={}*)
Bases: *shinken.objects.item.Item*

get_name()

id = 1

my_type = 'serviceescalation'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

class shinken.objects.serviceescalation.**Serviceescalations**(*items*)
Bases: *shinken.objects.item.Items*

explode(*escalations*)

inner_class
alias of *Serviceescalation*

name_property = ''

17.7.29 serviceextinfo Module

This is the main class for the Service ext info. In fact it's mainly about the configuration part. Parameters are merged in Service so it's no use in running part

class shinken.objects.serviceextinfo.**ServiceExtInfo**(*params={}*)

Bases: *shinken.objects.item.Item*

get_dbg_name()

get_full_name()

get_name()

host_name

icon_image

icon_image_alt

id = 1

imported_from

is_correct()

macros = {'SERVICENOTES': 'notes', 'SERVICEDESC': 'service_description', 'SERVICENOTESURL': 'notes_url', '

my_type = 'serviceextinfo'

name

notes

notes_url

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

service_description

use

class shinken.objects.serviceextinfo.**ServicesExtInfo**(*items*)

Bases: *shinken.objects.item.Items*

inner_class

alias of *ServiceExtInfo*

merge(*services*)

merge_extinfo(*service*, *extinfo*)

name_property = 'host_name'

17.7.30 servicegroup Module

class shinken.objects.servicegroup.**Servicegroup**(*params={}*)

Bases: *shinken.objects.itemgroup.Itemgroup*

get_name()

get_servicegroup_members()

get_services()

get_services_by_explosion(*servicegroups*)

id = 1

macros = {'SERVICEGROUPNOTES': 'notes', 'SERVICEGROUPALIAS': 'alias', 'SERVICEGROUPMEMBERS': 'm

my_type = 'servicegroup'

properties = {'use': <shinken.property.ListProp object at 0x7fa5c3ecdb90>, 'name': <shinken.property.StringProp ob

class shinken.objects.servicegroup.**Servicegroups**(*items*)

Bases: *shinken.objects.itemgroup.Itemgroups*

add_member(*cname*, *sgname*)

explode()

inner_class

alias of *Servicegroup*

linkify(*services*)

linkify_sg_by_srv(*services*)

name_property = 'servicegroup_name'

17.7.31 timeperiod Module

class shinken.objects.timeperiod.**Timeperiod**(*params={}*)

Bases: *shinken.objects.item.Item*

apply_inheritance()

check_and_log_activation_change()

check_exclude_rec()

clean_cache()

explode(*timeperiods*)

fill_data_brok_from(*data*, *brok_type*)

find_next_invalid_time_from_cache(*t*)

find_next_valid_time_from_cache(*t*)

get_initial_status_brok()

get_min_from_t(*t*)

get_name()

get_next_invalid_time_from_t(*t*)

get_next_valid_time_from_t(*t*)

get_not_in_min_from_t(*f*)

get_raw_import_values()

get_unresolved_properties_by_inheritance(*items*)

has(*prop*)

id = 1

is_correct()

is_time_valid(*t*)

linkify(*timeperiods*)

my_type = 'timeperiod'

properties = {'use': <shinken.property.StringProp object at 0x7fa5c3ee05d0>, 'name': <shinken.property.StringProp

resolve_daterange(*dateranges*, *entry*)

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.timeperiod.**Timeperiods**(*items*)
Bases: *shinken.objects.item.Items*

apply_inheritance()

explode()

inner_class
alias of *Timeperiod*

is_correct()

linkify()

name_property = 'timeperiod_name'

17.7.32 trigger Module

class shinken.objects.trigger.**Trigger**(*params={}*)
Bases: *shinken.objects.item.Item*

compile()

eval(*myself*, *ctx*)

get_name()

id = 1

my_type = 'trigger'

properties = {'trigger_name': <shinken.property.StringProp object at 0x7fa5c5d36510>, 'use': <shinken.property.List

running_properties = {'configuration_errors': <shinken.property.ListProp object at 0x7fa5c3ecdc50>, 'configuratio

class shinken.objects.trigger.**Triggers**(*items*)
Bases: *shinken.objects.item.Items*

compile()

create_trigger(*src*, *name*)

inner_class
alias of *Trigger*

load_file(*path*)

load_objects(*conf*)

name_property = 'trigger_name'

17.8 webui Package

17.8.1 `webui` Package

17.8.2 `bottlecore` Module

17.8.3 `bottlewebui` Module

CHAPTER 18

Indices and tables

- genindex
- modindex
- search

S

shinken.__init__, 13336
shinken.acknowledge, 13336
shinken.action, 13336
shinken.autoslots, 13336
shinken.basemodule, 13337
shinken.borg, 13338
shinken.brok, 13338
shinken.check, 13338
shinken.clients, 13392
shinken.clients.livestatus, 13393
shinken.clients.LSB, 13392
shinken.commandcall, 13339
shinken.comment, 13340
shinken.complexexpression, 13340
shinken.contactdowntime, 13340
shinken.daemons, 13394
shinken.daterange, 13341
shinken.db, 13342
shinken.db_mysql, 13343
shinken.db_oracle, 13343
shinken.db_sqlite, 13343
shinken.dependencynode, 13343
shinken.discovery, 13394
shinken.dispatcher, 13344
shinken.downtime, 13344
shinken.easter, 13344
shinken.eventhandler, 13346
shinken.graph, 13346
shinken.load, 13347
shinken.log, 13347
shinken.macroresolver, 13348
shinken.memoized, 13348
shinken.message, 13348
shinken.misc, 13394
shinken.misc.datamanager, 13394
shinken.misc.filter, 13395
shinken.misc.md5crypt, 13396
shinken.misc.perfdata, 13396
shinken.misc.sorter, 13396
shinken.misc.termcolor, 13396
shinken.modulesmanager, 13349
shinken.notification, 13349
shinken.objects, 13397
shinken.objects.businessimpactmodulation, 13397
shinken.objects.checkmodulation, 13397
shinken.objects.command, 13398
shinken.objects.contact, 13399
shinken.objects.contactgroup, 13399
shinken.objects.discoveryrule, 13400
shinken.objects.discoveryrun, 13400
shinken.objects.escalation, 13401
shinken.objects.host, 13401
shinken.objects.hostdependency, 13408
shinken.objects.hostescalation, 13408
shinken.objects.hostextinfo, 13409
shinken.objects.hostgroup, 13410
shinken.objects.item, 13410
shinken.objects.itemgroup, 13413
shinken.objects.macromodulation, 13413
shinken.objects.matchingitem, 13414
shinken.objects.module, 13414
shinken.objects.notificationway, 13414
shinken.objects.pack, 13415
shinken.objects.realm, 13415
shinken.objects.resultmodulation, 13416
shinken.objects.schedulingitem, 13417
shinken.objects.service, 13418
shinken.objects.servicedependency, 13425
shinken.objects.serviceescalation, 13425
shinken.objects.serviceextinfo, 13425
shinken.objects.servicegroup, 13426
shinken.objects.timeperiod, 13427
shinken.objects.trigger, 13428
shinken.property, 13351
shinken.singleton, 13352
shinken.sorteddict, 13352
shinken.util, 13354
shinken.webui, 13429
shinken.worker, 13355

A

ack_author (shinken.notification.Notification attribute), 13295, 13349
ack_data (shinken.notification.Notification attribute), 13295, 13350
Acknowledge (class in shinken.acknowledge), 13281, 13336
acknowledge_problem() (shinken.objects.item.Item method), 13318, 13374, 13410
acknowledgement (shinken.objects.host.Host attribute), 13309, 13365, 13402
acknowledgement (shinken.objects.service.Service attribute), 13326, 13382, 13418
acknowledgement_type (shinken.objects.host.Host attribute), 13309, 13365, 13402
acknowledgement_type (shinken.objects.service.Service attribute), 13326, 13382, 13418
act_depend_of (shinken.objects.host.Host attribute), 13309, 13365, 13402
act_depend_of (shinken.objects.service.Service attribute), 13326, 13382, 13418
act_depend_of_me (shinken.objects.host.Host attribute), 13309, 13365, 13402
act_depend_of_me (shinken.objects.service.Service attribute), 13326, 13382, 13418
Action (class in shinken.action), 13281, 13336
action_url (shinken.objects.host.Host attribute), 13310, 13365, 13402
action_url (shinken.objects.service.Service attribute), 13326, 13382, 13418
actions (shinken.objects.host.Host attribute), 13310, 13365, 13402
actions (shinken.objects.service.Service attribute), 13326, 13382, 13418
active_checks_enabled (shinken.objects.host.Host attribute), 13310, 13365, 13402
active_checks_enabled (shinken.objects.service.Service attribute), 13326, 13382, 13418
add() (shinken.objects.itemgroup.Itemgroups method), 13320, 13377, 13413
add_attempt() (shinken.objects.schedulingitem.SchedulingItem method), 13324, 13381, 13417
add_automatic_comment() (shinken.downtime.Downtime method), 13289, 13344
add_business_rule_act_dependency() (shinken.objects.host.Host method), 13310, 13365, 13402
add_business_rule_act_dependency() (shinken.objects.service.Service method), 13326, 13382, 13418
add_comment() (shinken.objects.item.Item method), 13318, 13374, 13410
add_contactgroup() (shinken.objects.contactgroup.Contactgroups method), 13308, 13363, 13400
add_downtime() (shinken.objects.item.Item method), 13318, 13374, 13410
add_edge() (shinken.graph.Graph method), 13292, 13346
add_escalation() (shinken.objects.escalation.Escalations method), 13309, 13365, 13401
add_flapping_change() (shinken.objects.schedulingitem.SchedulingItem method), 13324, 13381, 13417
add_host_act_dependency() (shinken.objects.host.Host method), 13310, 13366, 13402
add_host_chk_dependency() (shinken.objects.host.Host method), 13310, 13366, 13402
add_idletime() (shinken.worker.Worker method), 13300, 13355
add_member() (shinken.objects.contactgroup.Contactgroups method), 13308, 13363, 13400
add_member() (shinken.objects.hostgroup.Hostgroups method), 13318, 13374, 13410
add_member() (shinken.objects.servicegroup.Servicegroups method), 13334, 13390, 13427
add_node() (shinken.graph.Graph method), 13292, 13347
add_nodes() (shinken.graph.Graph method), 13292, 13347
add_service_act_dependency() (shinken.objects.service.Service method), 13326, 13382, 13419

add_service_chk_dependency() (shinken.objects.service.Service method), 13326, 13382, 13419
add_service_dependency() (shinken.objects.servicedependency.Servicedependencies method), 13332, 13388, 13425
add_service_link() (shinken.objects.host.Host method), 13310, 13366, 13402
add_string_member() (shinken.objects.itemgroup.Itemgroup method), 13320, 13376, 13413
add_string_member() (shinken.objects.realm.Realm method), 13323, 13379, 13415
address (shinken.objects.host.Host attribute), 13310, 13366, 13402
aggregation (shinken.objects.service.Service attribute), 13326, 13382, 13419
alias (shinken.objects.host.Host attribute), 13310, 13366, 13402
alive_then_spare_then_deads() (in module shinken.util), 13299, 13354
already_start_escalations (shinken.notification.Notification attribute), 13295, 13350
apache_md5_crypt() (in module shinken.misc.md5crypt), 13304, 13360, 13396
apply_change_recursive_pattern_change() (in module shinken.util), 13299, 13354
apply_dependencies() (shinken.objects.host.Hosts method), 13315, 13371, 13407
apply_dependencies() (shinken.objects.service.Services method), 13331, 13388, 13424
apply_implicit_inheritance() (shinken.objects.service.Services method), 13331, 13388, 13424
apply_inheritance() (shinken.objects.hostdependency.Hostdependencies method), 13316, 13372, 13408
apply_inheritance() (shinken.objects.item.Items method), 13319, 13375, 13411
apply_inheritance() (shinken.objects.service.Services method), 13331, 13388, 13424
apply_inheritance() (shinken.objects.servicedependency.Servicedependencies method), 13332, 13388, 13425
apply_inheritance() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
apply_inheritance() (shinken.objects.timeperiod.Timeperiods method), 13335, 13391, 13428
apply_partial_inheritance() (shinken.objects.item.Items method), 13319, 13375, 13412
args (shinken.commandcall.CommandCall attribute), 13284, 13339
attempt (shinken.objects.host.Host attribute), 13310, 13366, 13402
attempt (shinken.objects.service.Service attribute), 13326, 13382, 13419
AutoSlots (class in shinken.autoslots), 13281, 13336

B

BaseModule (class in shinken.basemodule), 13282, 13337
BoolProp (class in shinken.property), 13296, 13351
Borg (class in shinken.borg), 13283, 13338
Brok (class in shinken.brok), 13283, 13338
broks (shinken.objects.host.Host attribute), 13310, 13366, 13402
broks (shinken.objects.service.Service attribute), 13326, 13382, 13419
business_impact (shinken.objects.host.Host attribute), 13310, 13366, 13402
business_impact (shinken.objects.service.Service attribute), 13326, 13382, 13419
business_impact_modulations (shinken.objects.host.Host attribute), 13310, 13366, 13402
business_impact_modulations (shinken.objects.service.Service attribute), 13326, 13382, 13419
business_rule (shinken.objects.host.Host attribute), 13310, 13366, 13402
business_rule (shinken.objects.service.Service attribute), 13326, 13382, 13419
Businessimpactmodulation (class in shinken.objects.businessimpactmodulation), 13305, 13361, 13397
Businessimpactmodulations (class in shinken.objects.businessimpactmodulation), 13305, 13361, 13397

C

CalendarDaterange (class in shinken.daterange), 13286, 13341
call (shinken.commandcall.CommandCall attribute), 13284, 13339
cancel() (shinken.contactdowntime.ContactDowntime method), 13285, 13340
cancel() (shinken.downtime.Downtime method), 13289, 13344
CharProp (class in shinken.property), 13297, 13351
Check (class in shinken.check), 13283, 13338
check_activation() (shinken.contactdowntime.ContactDowntime method), 13285, 13340
check_alive() (shinken.dispatcher.Dispatcher method), 13289, 13344
check_alive_instances() (shinken.modulesmanager.ModulesManager method), 13294, 13349
check_and_log_activation_change() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
check_and_set_unreachability() (shinken.objects.schedulingitem.SchedulingItem

method), 13324, 13381, 13417
check_bad_dispatch() (shinken.dispatcher.Dispatcher method), 13289, 13344
check_command (shinken.objects.host.Host attribute), 13310, 13366, 13402
check_command (shinken.objects.service.Service attribute), 13326, 13382, 13419
check_dispatch() (shinken.dispatcher.Dispatcher method), 13289, 13344
check_exclude_rec() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
check_finished() (shinken.objects.discoveryrun.Discoveryrun method), 13308, 13364, 13400
check_flapping_recovery_notification (shinken.objects.host.Host attribute), 13310, 13366, 13402
check_flapping_recovery_notification (shinken.objects.service.Service attribute), 13326, 13383, 13419
check_for_expire_acknowledge() (shinken.objects.item.Item method), 13318, 13374, 13410
check_for_flexible_downtime() (shinken.objects.schedulingitem.SchedulingItem method), 13324, 13381, 13417
check_for_system_time_change() (shinken.worker.Worker method), 13300, 13355
check_freshness (shinken.objects.host.Host attribute), 13310, 13366, 13402
check_freshness (shinken.objects.service.Service attribute), 13326, 13383, 13419
check_interval (shinken.objects.host.Host attribute), 13310, 13366, 13402
check_interval (shinken.objects.service.Service attribute), 13326, 13383, 13419
check_period (shinken.objects.host.Host attribute), 13310, 13366, 13402
check_period (shinken.objects.service.Service attribute), 13326, 13383, 13419
check_time (shinken.check.Check attribute), 13283, 13338
check_time (shinken.eventhandler.EventHandler attribute), 13291, 13346
check_time (shinken.notification.Notification attribute), 13295, 13350
check_type (shinken.check.Check attribute), 13283, 13338
check_type (shinken.objects.host.Host attribute), 13310, 13366, 13402
check_type (shinken.objects.service.Service attribute), 13326, 13383, 13419
CheckModulation (class in shinken.objects.checkmodulation), 13305, 13361, 13397
CheckModulations (class in shinken.objects.checkmodulation), 13306, 13361, 13397
checkmodulations (shinken.objects.host.Host attribute), 13310, 13366, 13402
checkmodulations (shinken.objects.service.Service attribute), 13326, 13383, 13419
checks_in_progress (shinken.objects.host.Host attribute), 13310, 13366, 13402
checks_in_progress (shinken.objects.service.Service attribute), 13326, 13383, 13419
child_dependencies (shinken.objects.host.Host attribute), 13310, 13366, 13402
child_dependencies (shinken.objects.service.Service attribute), 13326, 13383, 13419
childs (shinken.objects.host.Host attribute), 13310, 13366, 13402
chk_depend_of (shinken.objects.host.Host attribute), 13310, 13366, 13402
chk_depend_of (shinken.objects.service.Service attribute), 13326, 13383, 13419
chk_depend_of_me (shinken.objects.host.Host attribute), 13310, 13366, 13402
chk_depend_of_me (shinken.objects.service.Service attribute), 13326, 13383, 13419
clean() (shinken.objects.item.Item method), 13318, 13374, 13410
clean() (shinken.objects.item.Items method), 13319, 13375, 13412
clean_cache() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
clear() (shinken.sorteddict.SortedDict method), 13298, 13353
clear_instances() (shinken.modulesmanager.ModulesManager method), 13294, 13349
clear_queues() (shinken.basemodule.BaseModule method), 13282, 13337
colored() (in module shinken.misc.termcolor), 13305, 13360, 13396
Command (class in shinken.objects.command), 13306, 13362, 13398
command (shinken.commandcall.CommandCall attribute), 13284, 13339
command (shinken.eventhandler.EventHandler attribute), 13291, 13346
command (shinken.notification.Notification attribute), 13295, 13350
command_call (shinken.notification.Notification attribute), 13295, 13350
command_line (shinken.objects.command.Command attribute), 13306, 13362, 13398
command_name (shinken.objects.command.Command attribute), 13306, 13362, 13398

CommandCall (class in shinken.commandcall), 13284, 13339
Commands (class in shinken.objects.command), 13306, 13362, 13398
Comment (class in shinken.comment), 13285, 13340
comments (shinken.objects.host.Host attribute), 13310, 13366, 13402
comments (shinken.objects.service.Service attribute), 13326, 13383, 13419
compensate_system_time_change() (shinken.objects.schedulingitem.SchedulingItem method), 13324, 13381, 13417
compile() (shinken.objects.trigger.Trigger method), 13335, 13392, 13428
compile() (shinken.objects.trigger.Triggers method), 13335, 13392, 13428
ComplexExpressionFactory (class in shinken.complexexpression), 13285, 13340
ComplexExpressionNode (class in shinken.complexexpression), 13285, 13340
compute_hash() (shinken.objects.item.Item method), 13318, 13374, 13410
compute_hash() (shinken.objects.item.Items method), 13319, 13375, 13412
CONFIGURATION (shinken.basemodule.ModulePhases attribute), 13283, 13337
configuration_errors (shinken.objects.host.Host attribute), 13310, 13366, 13402
configuration_errors (shinken.objects.service.Service attribute), 13326, 13383, 13419
configuration_warnings (shinken.objects.host.Host attribute), 13310, 13366, 13402
configuration_warnings (shinken.objects.service.Service attribute), 13326, 13383, 13419
connect() (shinken.clients.livestatus.LSSyncConnection method), 13302, 13357, 13393
connect_database() (shinken.db_mysql.DBMysql method), 13288, 13343
connect_database() (shinken.db_oracle.DBOracle method), 13288, 13343
connect_database() (shinken.db_sqlite.DBSqlite method), 13288, 13343
consume_result() (shinken.objects.schedulingitem.SchedulingItem method), 13324, 13381, 13417
Contact (class in shinken.objects.contact), 13307, 13362, 13399
contact (shinken.notification.Notification attribute), 13295, 13350
contact_groups (shinken.objects.host.Host attribute), 13310, 13366, 13402
contact_groups (shinken.objects.service.Service attribute), 13326, 13383, 13419
contact_name (shinken.notification.Notification attribute), 13295, 13350
ContactDowntime (class in shinken.contactdowntime), 13285, 13340
Contactgroup (class in shinken.objects.contactgroup), 13307, 13363, 13399
Contactgroups (class in shinken.objects.contactgroup), 13308, 13363, 13400
Contacts (class in shinken.objects.contact), 13307, 13363, 13399
contacts (shinken.objects.host.Host attribute), 13310, 13366, 13402
contacts (shinken.objects.service.Service attribute), 13326, 13383, 13419
contacts_notified (shinken.notification.Notification attribute), 13295, 13350
copy() (shinken.objects.item.Item method), 13318, 13374, 13410
copy() (shinken.sorteddict.SortedDict method), 13298, 13353
copy_create_service_from_another() (shinken.objects.service.Services method), 13331, 13388, 13424
copy_shell() (shinken.check.Check method), 13283, 13338
copy_shell() (shinken.eventhandler.EventHandler method), 13291, 13346
copy_shell() (shinken.notification.Notification method), 13295, 13350
copy_shell() (shinken.objects.itemgroup.Itemgroup method), 13320, 13376, 13413
count_brokers() (shinken.objects.realm.Realm method), 13323, 13379, 13415
count_pollers() (shinken.objects.realm.Realm method), 13323, 13379, 13415
count_reactionners() (shinken.objects.realm.Realm method), 13323, 13379, 13415
count_receivers() (shinken.objects.realm.Realm method), 13323, 13379, 13415
cprint() (in module shinken.log), 13293, 13348
cprint() (in module shinken.misc.termcolor), 13305, 13360, 13397
create_business_rules() (shinken.objects.host.Hosts method), 13315, 13371, 13407
create_business_rules() (shinken.objects.schedulingitem.SchedulingItem method), 13324, 13381, 13417
create_business_rules() (shinken.objects.service.Services method), 13331, 13388, 13424
create_business_rules_dependencies() (shinken.objects.host.Hosts method), 13315, 13371, 13407
create_business_rules_dependencies() (shinken.objects.schedulingitem.SchedulingItem method), 13324, 13381, 13417
create_business_rules_dependencies() (shinken.objects.service.Services method),

13331, 13388, 13424
create_insert_query() (shinken.db.DB method), 13287, 13342
create_notifications() (shinken.objects.schedulingitem.SchedulingItem method), 13324, 13381, 13417
create_pack() (shinken.objects.pack.Packs method), 13322, 13379, 13415
create_queues() (shinken.basemodule.BaseModule method), 13282, 13337
create_reversed_list() (shinken.objects.item.Items method), 13319, 13375, 13412
create_reversed_list() (shinken.objects.service.Services method), 13331, 13388, 13424
create_tpl_list() (shinken.objects.item.Items method), 13319, 13375, 13412
create_trigger() (shinken.objects.trigger.Triggers method), 13335, 13392, 13428
create_update_query() (shinken.db.DB method), 13287, 13342
creation_time (shinken.notification.Notification attribute), 13295, 13350
CRITICAL (shinken.log.Log attribute), 13292, 13347
critical() (shinken.log.Log method), 13292, 13347
current_event_id (shinken.objects.host.Host attribute), 13310, 13366, 13403
current_event_id (shinken.objects.schedulingitem.SchedulingItem attribute), 13324, 13381, 13417
current_event_id (shinken.objects.service.Service attribute), 13327, 13383, 13419
current_notification_id (shinken.objects.host.Host attribute), 13310, 13366, 13403
current_notification_id (shinken.objects.service.Service attribute), 13327, 13383, 13419
current_notification_number (shinken.objects.host.Host attribute), 13310, 13366, 13403
current_notification_number (shinken.objects.service.Service attribute), 13327, 13383, 13419
current_problem_id (shinken.objects.host.Host attribute), 13310, 13366, 13403
current_problem_id (shinken.objects.schedulingitem.SchedulingItem attribute), 13325, 13381, 13417
current_problem_id (shinken.objects.service.Service attribute), 13327, 13383, 13419
custom_views (shinken.objects.host.Host attribute), 13310, 13366, 13403
custom_views (shinken.objects.service.Service attribute), 13327, 13383, 13419
customs (shinken.objects.host.Host attribute), 13311, 13366, 13403
customs (shinken.objects.service.Service attribute), 13327, 13383, 13419

D

dark() (in module shinken.easter), 13289, 13344
DataManager (class in shinken.misc.datamanager), 13303, 13358, 13394
Daterange (class in shinken.daterange), 13286, 13341
DB (class in shinken.db), 13287, 13342
DBMysql (class in shinken.db_mysql), 13288, 13343
DBOracle (class in shinken.db_oracle), 13288, 13343
DBSqlite (class in shinken.db_sqlite), 13288, 13343
DEBUG (shinken.log.Log attribute), 13292, 13347
debug() (shinken.log.Log method), 13292, 13347
default_value (shinken.objects.service.Service attribute), 13327, 13383, 13419
del_automatic_comment() (shinken.downtime.Downtime method), 13289, 13344
del_comment() (shinken.objects.item.Item method), 13318, 13374, 13411
del_downtime() (shinken.objects.item.Item method), 13318, 13374, 13411
del_host_act_dependency() (shinken.objects.host.Host method), 13311, 13366, 13403
delete_hostsdep_by_id() (shinken.objects.hostdependency.Hostdependencie method), 13316, 13372, 13408
delete_services_by_id() (shinken.objects.service.Services method), 13331, 13388, 13424
delete_servicesdep_by_id() (shinken.objects.servicedependency.Servicedependencies method), 13332, 13388, 13425
dep_check (shinken.check.Check attribute), 13283, 13338
depend_on (shinken.check.Check attribute), 13283, 13338
DependencyNode (class in shinken.dependencynode), 13288, 13343
DependencyNodeFactory (class in shinken.dependencynode), 13288, 13343
deregister_a_problem() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
dfs_get_all_childs() (shinken.graph.Graph method), 13292, 13347
dfs_loop_search() (shinken.graph.Graph method), 13292, 13347
dictdoc() (in module shinken.sorteddict), 13299, 13353
disable_active_checks() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
Discoveryrule (class in shinken.objects.discoveryrule), 13308, 13364, 13400
Discoveryrules (class in shinken.objects.discoveryrule), 13308, 13364, 13400
Discoveryrun (class in shinken.objects.discoveryrun), 13308, 13364, 13400
Discoveryruns (class in shinken.objects.discoveryrun), 13309, 13364, 13401

dispatch() (shinken.dispatcher.Dispatcher method), 13289, 13344
Dispatcher (class in shinken.dispatcher), 13289, 13344
display_name (shinken.objects.host.Host attribute), 13311, 13366, 13403
display_name (shinken.objects.service.Service attribute), 13327, 13383, 13419
do_check_freshness() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
do_connect() (shinken.clients.livestatus.LSAsynConnection method), 13301, 13356, 13393
do_i_raise_dependency() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
do_loop_turn() (shinken.basemodule.BaseModule method), 13282, 13337
do_read() (shinken.clients.livestatus.LSAsynConnection method), 13301, 13356, 13393
do_stop() (shinken.basemodule.BaseModule method), 13282, 13337
do_work() (shinken.worker.Worker method), 13300, 13355
Downtime (class in shinken.downtime), 13289, 13344
downtimes (shinken.objects.host.Host attribute), 13311, 13366, 13403
downtimes (shinken.objects.service.Service attribute), 13327, 13383, 13419
DummyCommand (class in shinken.objects.command), 13306, 13362, 13398
DummyCommandCall (class in shinken.commandcall), 13285, 13340
duplicate() (shinken.objects.service.Service method), 13327, 13383, 13419
duplicate_foreach (shinken.objects.service.Service attribute), 13327, 13383, 13419
duration_sec (shinken.objects.host.Host attribute), 13311, 13367, 13403
duration_sec (shinken.objects.service.Service attribute), 13327, 13383, 13419

E

early_timeout (shinken.objects.host.Host attribute), 13311, 13367, 13403
early_timeout (shinken.objects.service.Service attribute), 13327, 13383, 13420
end_time (shinken.notification.Notification attribute), 13295, 13350
end_time (shinken.objects.host.Host attribute), 13311, 13367, 13403
end_time (shinken.objects.service.Service attribute), 13327, 13383, 13420
enter() (shinken.contactdowntime.ContactDowntime method), 13286, 13340
enter() (shinken.downtime.Downtime method), 13289, 13344
env (shinken.check.Check attribute), 13283, 13338
env (shinken.eventhandler.EventHandler attribute), 13291, 13346
env (shinken.notification.Notification attribute), 13295, 13350
epilogue() (in module shinken.easter), 13290, 13345
ERROR (shinken.log.Log attribute), 13292, 13347
error() (shinken.log.Log method), 13292, 13347
escalated (shinken.notification.Notification attribute), 13295, 13350
Escalation (class in shinken.objects.escalation), 13309, 13365, 13401
Escalations (class in shinken.objects.escalation), 13309, 13365, 13401
escalations (shinken.objects.host.Host attribute), 13311, 13367, 13403
escalations (shinken.objects.service.Service attribute), 13327, 13383, 13420
eval() (shinken.objects.trigger.Trigger method), 13335, 13392, 13428
eval_cor_pattern() (shinken.complexexpression.ComplexExpressionFactory method), 13285, 13340
eval_cor_pattern() (shinken.dependencynode.DependencyNodeFactory method), 13289, 13343
eval_triggers() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
evaluate_hostgroup_expression() (shinken.objects.item.Items method), 13319, 13375, 13412
event_handler (shinken.objects.host.Host attribute), 13311, 13367, 13403
event_handler (shinken.objects.service.Service attribute), 13327, 13383, 13420
event_handler_enabled (shinken.objects.host.Host attribute), 13311, 13367, 13403
event_handler_enabled (shinken.objects.service.Service attribute), 13327, 13383, 13420
EventHandler (class in shinken.eventhandler), 13291, 13346
exec_command() (shinken.clients.livestatus.LSAsynConnection method), 13301, 13356, 13393
exec_command() (shinken.clients.livestatus.LSSyncConnection method), 13302, 13357, 13393
execute__() (shinken.action.Action method), 13281, 13336
execute_query() (shinken.db_mysql.DBMysql method), 13288, 13343
execute_query() (shinken.db_oracle.DBOracle method), 13288, 13343
execute_query() (shinken.db_sqlite.DBSqlite method), 13288, 13343

execution_time (shinken.check.Check attribute), 13283, 13338
execution_time (shinken.eventhandler.EventHandler attribute), 13291, 13346
execution_time (shinken.notification.Notification attribute), 13295, 13350
execution_time (shinken.objects.host.Host attribute), 13311, 13367, 13403
execution_time (shinken.objects.service.Service attribute), 13327, 13383, 13420
exit() (shinken.contactdowntime.ContactDowntime method), 13286, 13340
exit() (shinken.downtime.Downtime method), 13289, 13344
exit_status (shinken.check.Check attribute), 13283, 13338
exit_status (shinken.eventhandler.EventHandler attribute), 13291, 13346
exit_status (shinken.notification.Notification attribute), 13295, 13350
expand_with_macros() (in module shinken.util), 13299, 13354
expand_xy_pattern() (in module shinken.util), 13299, 13354
expect_file_dirs() (in module shinken.util), 13299, 13354
explode() (shinken.objects.contact.Contacts method), 13307, 13363, 13399
explode() (shinken.objects.contactgroup.Contactgroups method), 13308, 13363, 13400
explode() (shinken.objects.escalation.Escalations method), 13309, 13365, 13401
explode() (shinken.objects.host.Hosts method), 13315, 13371, 13408
explode() (shinken.objects.hostdependency.Hostdependencies method), 13316, 13372, 13408
explode() (shinken.objects.hostescalation.Hostescalations method), 13316, 13372, 13409
explode() (shinken.objects.hostgroup.Hostgroups method), 13318, 13374, 13410
explode() (shinken.objects.module.Modules method), 13321, 13378, 13414
explode() (shinken.objects.realm.Realms method), 13323, 13380, 13416
explode() (shinken.objects.service.Services method), 13331, 13388, 13424
explode() (shinken.objects.servicedependency.Servicedependencies method), 13332, 13388, 13425
explode() (shinken.objects.serviceescalation.Serviceescalations method), 13333, 13389, 13425
explode() (shinken.objects.servicegroup.Servicegroups method), 13334, 13390, 13427
explode() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
explode() (shinken.objects.timeperiod.Timeperiods method), 13335, 13391, 13428
explode_contact_groups_into_contacts() (shinken.objects.item.Items method), 13319, 13375, 13412
explode_host_groups_into_hosts() (shinken.objects.item.Items method), 13319, 13376, 13412
explode_hostgroup() (shinken.objects.servicedependency.Servicedependenc method), 13332, 13388, 13425
explode_trigger_string_into_triggers() (shinken.objects.item.Item method), 13318, 13374, 13411
explode_trigger_string_into_triggers() (shinken.objects.item.Items method), 13319, 13376, 13412

F

failure_prediction_enabled (shinken.objects.host.Host attribute), 13311, 13367, 13403
failure_prediction_enabled (shinken.objects.service.Service attribute), 13327, 13383, 13420
fetchone() (shinken.db.DB method), 13287, 13342
FileQueryGenerator (class in shinken.clients.LSB), 13301, 13356, 13392
fill_broker_with_poller_reactionner_links() (shinken.objects.realm.Realm method), 13323, 13379, 13415
fill_daddy_dependency() (shinken.objects.service.Service method), 13327, 13383, 13420
fill_data_brok_from() (shinken.downtime.Downtime method), 13289, 13344
fill_data_brok_from() (shinken.notification.Notification method), 13295, 13350
fill_data_brok_from() (shinken.objects.command.Command method), 13306, 13362, 13398
fill_data_brok_from() (shinken.objects.item.Item method), 13318, 13374, 13411
fill_data_brok_from() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
fill_default() (shinken.objects.item.Item method), 13318, 13374, 13411
fill_default() (shinken.objects.item.Items method), 13319, 13376, 13412
fill_default() (shinken.objects.itemgroup.Itemgroup method), 13320, 13376, 13413
fill_default() (shinken.objects.itemgroup.Itemgroups method), 13321, 13377, 13413
fill_parents_dependency() (shinken.objects.host.Host method), 13311, 13367, 13403
fill_potential_brokers() (shinken.objects.realm.Realm method), 13323, 13379, 13415

fill_potential_pollers() (shinken.objects.realm.Realm method), 13323, 13379, 13415
fill_potential_reactionners() (shinken.objects.realm.Realm method), 13323, 13379, 13415
fill_potential_receivers() (shinken.objects.realm.Realm method), 13323, 13379, 13415
fill_predictive_missing_parameters() (shinken.objects.host.Host method), 13311, 13367, 13403
fill_predictive_missing_parameters() (shinken.objects.host.Hosts method), 13315, 13371, 13408
find_by_name() (shinken.objects.item.Items method), 13319, 13376, 13412
find_day_by_offset() (in module shinken.daterange), 13287, 13342
find_day_by_weekday_offset() (in module shinken.daterange), 13287, 13342
find_hosts_that_use_template() (shinken.objects.host.Hosts method), 13315, 13371, 13408
find_id_by_name() (shinken.objects.item.Items method), 13320, 13376, 13412
find_next_invalid_time_from_cache() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
find_next_valid_time_from_cache() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
find_object() (shinken.complexexpression.ComplexExpressionFactory method), 13285, 13340
find_object() (shinken.dependencynode.DependencyNodeFactory method), 13289, 13344
find_service_by_name() (shinken.objects.host.Host method), 13311, 13367, 13403
find_srv_by_name_and_hostname() (shinken.objects.service.Services method), 13332, 13388, 13424
find_srv_id_by_name_and_hostname() (shinken.objects.service.Services method), 13332, 13388, 13424
find_tpl_by_name() (shinken.objects.item.Items method), 13320, 13376, 13412
first_notification_delay (shinken.objects.host.Host attribute), 13311, 13367, 13403
first_notification_delay (shinken.objects.service.Service attribute), 13327, 13383, 13420
flap_detection_enabled (shinken.objects.host.Host attribute), 13311, 13367, 13403
flap_detection_enabled (shinken.objects.service.Service attribute), 13327, 13384, 13420
flap_detection_options (shinken.objects.host.Host attribute), 13311, 13367, 13403
flap_detection_options (shinken.objects.service.Service attribute), 13327, 13384, 13420
flapping_changes (shinken.objects.host.Host attribute), 13311, 13367, 13403
flapping_changes (shinken.objects.service.Service attribute), 13327, 13384, 13420
flapping_comment_id (shinken.objects.host.Host attribute), 13311, 13367, 13403
flapping_comment_id (shinken.objects.service.Service attribute), 13327, 13384, 13420
FloatProp (class in shinken.property), 13297, 13351
format_t_into_dhms_format() (in module shinken.util), 13299, 13354
freshness_threshold (shinken.objects.host.Host attribute), 13311, 13367, 13403
freshness_threshold (shinken.objects.service.Service attribute), 13327, 13384, 13420
from_bool_to_int() (in module shinken.util), 13299, 13354
from_bool_to_string() (in module shinken.util), 13299, 13354
from_float_to_int() (in module shinken.util), 13299, 13354
from_list_to_split() (in module shinken.util), 13299, 13354
from_trigger (shinken.check.Check attribute), 13283, 13338
fromkeys() (shinken.sorteddict.SortedDict class method), 13298, 13353

G

get() (shinken.clients.livestatus.Query method), 13302, 13357, 13394
get() (shinken.clients.LSB.QueryGenerator method), 13301, 13356, 13392
get() (shinken.clients.LSB.SimpleQueryGenerator method), 13301, 13356, 13393
get_accessibility_packs() (shinken.graph.Graph method), 13292, 13347
get_ack_author_name() (shinken.objects.host.Host method), 13311, 13367, 13403
get_ack_author_name() (shinken.objects.service.Service method), 13327, 13384, 13420
get_ack_comment() (shinken.objects.host.Host method), 13311, 13367, 13403
get_ack_comment() (shinken.objects.service.Service method), 13327, 13384, 13420
get_all_hosts_and_services() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13394
get_all_impacts() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13394
get_all_plus_and_delete() (shinken.objects.item.Item method), 13318, 13374, 13411

get_all_problems() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13394
get_all_subs_pollers() (shinken.objects.realm.Realm method), 13323, 13379, 13416
get_all_subs_reactionners() (shinken.objects.realm.Realm method), 13323, 13379, 13416
get_brokers() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13394
get_business_parents() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13394
get_check_command() (shinken.objects.checkmodulation.CheckModulation method), 13305, 13361, 13397
get_check_command() (shinken.objects.host.Host method), 13311, 13367, 13403
get_check_command() (shinken.objects.service.Service method), 13327, 13384, 13420
get_check_result_brok() (shinken.objects.item.Item method), 13318, 13374, 13411
get_coffee() (in module shinken.easter), 13290, 13345
get_command_and_args() (shinken.commandcall.CommandCall method), 13284, 13339
get_contact() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13394
get_contactgroup_members() (shinken.objects.contactgroup.Contactgroup method), 13307, 13363, 13399
get_contacts() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13394
get_contacts() (shinken.objects.contactgroup.Contactgroup method), 13307, 13363, 13399
get_contacts_by_explosion() (shinken.objects.contactgroup.Contactgroup method), 13307, 13363, 13399
get_customs_keys() (in module shinken.util), 13299, 13354
get_customs_properties_by_inheritance() (shinken.objects.item.Item method), 13318, 13375, 13411
get_customs_values() (in module shinken.util), 13299, 13354
get_data() (shinken.message.Message method), 13294, 13348
get_data_for_checks() (shinken.objects.host.Host method), 13311, 13367, 13403
get_data_for_checks() (shinken.objects.service.Service method), 13327, 13384, 13420
get_data_for_event_handler() (shinken.objects.host.Host method), 13311, 13367, 13403
get_data_for_event_handler() (shinken.objects.service.Service method), 13327, 13384, 13420
get_data_for_notifications() (shinken.objects.host.Host method), 13311, 13367, 13403
get_data_for_notifications() (shinken.objects.service.Service method), 13327, 13384, 13420
get_day() (in module shinken.util), 13299, 13354
get_dbg_name() (shinken.objects.host.Host method), 13311, 13367, 13403
get_dbg_name() (shinken.objects.hostextinfo.HostExtInfo method), 13317, 13373, 13409
get_dbg_name() (shinken.objects.service.Service method), 13327, 13384, 13420
get_dbg_name() (shinken.objects.serviceextinfo.ServiceExtInfo method), 13333, 13389, 13426
get_default() (shinken.objects.realm.Realms method), 13323, 13380, 13416
get_duration() (shinken.objects.host.Host method), 13311, 13367, 13403
get_duration() (shinken.objects.service.Service method), 13327, 13384, 13420
get_duration_sec() (shinken.objects.host.Host method), 13311, 13367, 13403
get_duration_sec() (shinken.objects.service.Service method), 13327, 13384, 13420
get_end_of_day() (in module shinken.util), 13299, 13354
get_env_macros() (shinken.macroresolver.MacroResolver method), 13293, 13348
get_escalable_contacts() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
get_event_handlers() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
get_external_from_queues() (shinken.modulesmanager.ModulesManager method), 13294, 13349
get_external_instances() (shinken.modulesmanager.ModulesManager method), 13294, 13349
get_external_to_queues() (shinken.modulesmanager.ModulesManager method), 13294, 13349
get_first_sec_out_from_morning() (shinken.daterange.Timerange method), 13287, 13342
get_from() (shinken.message.Message method), 13294, 13348
get_full_name() (shinken.objects.host.Host method), 13311, 13367, 13403
get_full_name() (shinken.objects.hostextinfo.HostExtInfo method), 13317, 13373, 13409
get_full_name() (shinken.objects.service.Service method), 13327, 13384, 13420
get_full_name() (shinken.objects.serviceextinfo.ServiceExtInfo method), 13333, 13389, 13426
get_groupnames() (shinken.objects.service.Service method), 13328, 13384, 13420

get_host() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13395
get_host_tags() (shinken.objects.host.Host method), 13311, 13367, 13404
get_host_tags() (shinken.objects.service.Service method), 13328, 13384, 13420
get_host_tags_sorted() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13395
get_hostgroup() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13395
get_hostgroup_members() (shinken.objects.hostgroup.Hostgroup method), 13317, 13373, 13410
get_hostgroups() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13395
get_hostgroups() (shinken.objects.host.Host method), 13311, 13367, 13404
get_hostgroups() (shinken.objects.service.Service method), 13328, 13384, 13420
get_hostgroups_sorted() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13395
get_hosts() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13395
get_hosts() (shinken.objects.hostgroup.Hostgroup method), 13317, 13373, 13410
get_hosts_by_explosion() (shinken.objects.hostgroup.Hostgroup method), 13317, 13373, 13410
get_id() (shinken.check.Check method), 13283, 13338
get_id() (shinken.eventhandler.EventHandler method), 13291, 13346
get_id() (shinken.notification.Notification method), 13295, 13350
get_important_elements() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13395
get_important_impacts() (shinken.misc.datamanager.DataManager method), 13303, 13358, 13395
get_initial_status_brok() (shinken.downtime.Downtime method), 13289, 13344
get_initial_status_brok() (shinken.notification.Notification method), 13295, 13350
get_initial_status_brok() (shinken.objects.command.Command method), 13306, 13362, 13398
get_initial_status_brok() (shinken.objects.item.Item method), 13318, 13375, 13411
get_initial_status_brok() (shinken.objects.itemgroup.Itemgroup method), 13320, 13376, 13413
get_initial_status_brok() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
get_instances() (shinken.modulesmanager.ModulesManager method), 13294, 13349
get_internal_instances() (shinken.modulesmanager.ModulesManager method), 13294, 13349
get_key_value_sequence() (in module shinken.util), 13299, 13354
get_len_overall_state() (shinken.misc.datamanager.DataManager method), 13303, 13359, 13395
get_level() (shinken.log.Log method), 13292, 13347
get_level_id() (shinken.log.Log static method), 13292, 13347
get_load() (shinken.load.Load method), 13292, 13347
get_members_by_name() (shinken.objects.contactgroup.Contactgroups method), 13308, 13363, 13400
get_members_by_name() (shinken.objects.hostgroup.Hostgroups method), 13318, 13374, 13410
get_members_by_name() (shinken.objects.itemgroup.Itemgroups method), 13321, 13377, 13413
get_members_by_name() (shinken.objects.realm.Realms method), 13324, 13380, 13416
get_min_from_t() (shinken.daterange.Daterange method), 13286, 13341
get_min_from_t() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
get_min_sec_from_morning() (shinken.daterange.Daterange method), 13286, 13341
get_min_sec_out_from_morning() (shinken.daterange.Daterange method), 13286, 13341
get_month_by_id() (shinken.daterange.Daterange class method), 13286, 13341
get_month_id() (shinken.daterange.Daterange class method), 13286, 13341
get_name() (shinken.basemodule.BaseModule method), 13282, 13337
get_name() (shinken.commandcall.CommandCall method), 13284, 13339
get_name() (shinken.objects.businessimpactmodulation.Businessimpactmod method), 13305, 13361, 13397
get_name() (shinken.objects.checkmodulation.CheckModulation method), 13305, 13361, 13397
get_name() (shinken.objects.command.Command method), 13306, 13362, 13398
get_name() (shinken.objects.contact.Contact method), 13307, 13362, 13399
get_name() (shinken.objects.contactgroup.Contactgroup method), 13307, 13363, 13399
get_name() (shinken.objects.discoveryrule.Discoveryrule method), 13308, 13364, 13400
get_name() (shinken.objects.discoveryrun.Discoveryrun method), 13308, 13364, 13400
get_name() (shinken.objects.escalation.Escalation method), 13309, 13365, 13401

get_name() (shinken.objects.host.Host method), 13311, 13367, 13404
get_name() (shinken.objects.hostdependency.Hostdependency method), 13316, 13372, 13408
get_name() (shinken.objects.hostescalation.Hostescalation method), 13316, 13372, 13408
get_name() (shinken.objects.hostextinfo.HostExtInfo method), 13317, 13373, 13409
get_name() (shinken.objects.hostgroup.Hostgroup method), 13317, 13373, 13410
get_name() (shinken.objects.macromodulation.MacroModulation method), 13321, 13377, 13413
get_name() (shinken.objects.module.Module method), 13321, 13377, 13414
get_name() (shinken.objects.notificationway.NotificationWay method), 13322, 13378, 13414
get_name() (shinken.objects.pack.Pack method), 13322, 13379, 13415
get_name() (shinken.objects.realm.Realm method), 13323, 13379, 13416
get_name() (shinken.objects.resultmodulation.Resultmodulation method), 13324, 13380, 13416
get_name() (shinken.objects.service.Service method), 13328, 13384, 13420
get_name() (shinken.objects.servicedependency.Servicedependency method), 13332, 13389, 13425
get_name() (shinken.objects.serviceescalation.Serviceescalation method), 13332, 13389, 13425
get_name() (shinken.objects.serviceextinfo.ServiceExtInfo method), 13333, 13389, 13426
get_name() (shinken.objects.servicegroup.Servicegroup method), 13334, 13390, 13426
get_name() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
get_name() (shinken.objects.trigger.Trigger method), 13335, 13392, 13428
get_nb_all_problems() (shinken.misc.datamanager.DataManager method), 13303, 13359, 13395
get_nb_elements() (shinken.misc.datamanager.DataManager method), 13303, 13359, 13395
get_nb_impacts() (shinken.misc.datamanager.DataManager method), 13303, 13359, 13395
get_nb_of_must_have_satellites() (shinken.objects.realm.Realm method), 13323, 13379, 13416
get_nb_problems() (shinken.misc.datamanager.DataManager method), 13303, 13359, 13395
get_new_checks() (shinken.worker.Worker method), 13300, 13355
get_next_future_timerange_invalid() (shinken.daterange.Daterange method), 13286, 13341
get_next_future_timerange_valid() (shinken.daterange.Daterange method), 13286, 13341
get_next_invalid_day() (shinken.daterange.Daterange method), 13286, 13341
get_next_invalid_time_from_t() (shinken.daterange.Daterange method), 13286, 13341
get_next_invalid_time_from_t() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
get_next_notif_time() (shinken.objects.escalation.Escalation method), 13309, 13365, 13401
get_next_notification_time() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
get_next_schedule_brok() (shinken.objects.item.Item method), 13318, 13375, 13411
get_next_valid_day() (shinken.daterange.Daterange method), 13286, 13341
get_next_valid_time_from_t() (shinken.daterange.Daterange method), 13286, 13341
get_next_valid_time_from_t() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
get_not_in_min_from_t() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
get_notification_commands() (shinken.objects.contact.Contact method), 13307, 13362, 13399
get_notification_commands() (shinken.objects.notificationway.NotificationWay method), 13322, 13378, 13414
get_obj_full_name() (in module shinken.util), 13299, 13354
get_obj_name() (in module shinken.util), 13299, 13354
get_obj_name_two_args_and_void() (in module shinken.util), 13299, 13354
get_obsessive_compulsive_processor_command() (shinken.objects.host.Host method), 13311, 13367, 13404
get_obsessive_compulsive_processor_command() (shinken.objects.service.Service method), 13328, 13384, 13420
get_output() (shinken.objects.discoveryrun.Discoveryrun method), 13308, 13364, 13400
get_outputs() (shinken.eventhandler.EventHandler method), 13291, 13346
get_overall_it_state() (shinken.misc.datamanager.DataManager method), 13303, 13359, 13395
get_overall_state() (shinken.misc.datamanager.DataManager method), 13303, 13359, 13395
get_per_hosts_state() (shinken.misc.datamanager.DataManager method), 13303, 13359, 13395

get_per_service_state() (shinken.misc.datamanager.DataManager method), 13303, 13359, 13395
get_perfdata_command() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
get_plus_and_delete() (shinken.objects.item.Item method), 13318, 13375, 13411
get_pollers() (shinken.misc.datamanager.DataManager method), 13303, 13359, 13395
get_potential_satellites_by_type() (shinken.objects.realm.Realm method), 13323, 13379, 13416
get_problems_time_sorted() (shinken.misc.datamanager.DataManager method), 13303, 13359, 13395
get_program_start() (shinken.misc.datamanager.DataManager method), 13304, 13359, 13395
get_property_by_inheritance() (shinken.objects.item.Item method), 13319, 13375, 13411
get_property_value_for_brok() (shinken.objects.item.Item method), 13319, 13375, 13411
get_query() (shinken.clients.livestatus.LSAsynConnection method), 13301, 13356, 13393
get_raw_import_values() (shinken.objects.item.Item method), 13319, 13375, 13411
get_raw_import_values() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
get_reactionners() (shinken.misc.datamanager.DataManager method), 13304, 13359, 13395
get_realm() (shinken.misc.datamanager.DataManager method), 13304, 13359, 13395
get_realm() (shinken.objects.host.Host method), 13311, 13367, 13404
get_realm() (shinken.objects.service.Service method), 13328, 13384, 13420
get_realm_members() (shinken.objects.realm.Realm method), 13323, 13379, 13416
get_realms() (shinken.misc.datamanager.DataManager method), 13304, 13359, 13395
get_realms() (shinken.objects.realm.Realm method), 13323, 13379, 13416
get_realms_by_explosion() (shinken.objects.realm.Realm method), 13323, 13379, 13416
get_receivers() (shinken.misc.datamanager.DataManager method), 13304, 13359, 13395
get_return_from() (shinken.check.Check method), 13284, 13338
get_return_from() (shinken.eventhandler.EventHandler method), 13291, 13346
get_return_from() (shinken.notification.Notification method), 13295, 13350
get_returns() (shinken.clients.livestatus.LSAsynConnection method), 13301, 13356, 13393
get_reverse_state() (shinken.dependencynode.DependencyNode method), 13288, 13343
get_satellites_links_for_scheduler() (shinken.objects.realm.Realm method), 13323, 13379, 13416
get_satellties_by_type() (shinken.objects.realm.Realm method), 13323, 13380, 13416
get_scheduler_ordered_list() (shinken.dispatcher.Dispatcher method), 13289, 13344
get_schedulers() (shinken.misc.datamanager.DataManager method), 13304, 13359, 13395
get_sec_from_morning() (in module shinken.util), 13299, 13354
get_sec_from_morning() (shinken.daterange.Timerange method), 13287, 13342
get_service() (shinken.misc.datamanager.DataManager method), 13304, 13359, 13395
get_servicegroup_members() (shinken.objects.servicegroup.Servicegroup method), 13334, 13390, 13426
get_services() (shinken.misc.datamanager.DataManager method), 13304, 13359, 13395
get_services() (shinken.objects.servicegroup.Servicegroup method), 13334, 13390, 13426
get_services_by_explosion() (shinken.objects.servicegroup.Servicegroup method), 13334, 13390, 13426
get_start_and_end_time() (shinken.daterange.CalendarDaterange method), 13286, 13341
get_start_and_end_time() (shinken.daterange.Daterange method), 13286, 13341
get_start_and_end_time() (shinken.daterange.MonthDateDaterange method), 13287, 13341
get_start_and_end_time() (shinken.daterange.MonthDayDaterange method), 13287, 13342
get_start_and_end_time() (shinken.daterange.MonthWeekDayDaterange method), 13287, 13342
get_start_and_end_time() (shinken.daterange.StandardDaterange method), 13287, 13342
get_start_and_end_time() (shinken.daterange.WeekDayDaterange method), 13287, 13342
get_start_of_day() (in module shinken.util), 13299, 13354

get_state() (shinken.dependencynode.DependencyNode method), 13288, 13343
get_templates() (shinken.objects.item.Item method), 13319, 13375, 13411
get_time_to_orphanage() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
get_total_services() (shinken.objects.host.Host method), 13312, 13367, 13404
get_total_services_critical() (shinken.objects.host.Host method), 13312, 13367, 13404
get_total_services_ok() (shinken.objects.host.Host method), 13312, 13367, 13404
get_total_services_unknown() (shinken.objects.host.Host method), 13312, 13367, 13404
get_total_services_warning() (shinken.objects.host.Host method), 13312, 13368, 13404
get_type() (shinken.message.Message method), 13294, 13348
get_unresolved_properties_by_inheritance() (shinken.objects.timeperiod.Timeperiod method), 13334, 13391, 13427
get_update_status_brok() (shinken.objects.item.Item method), 13319, 13375, 13411
get_wday() (in module shinken.util), 13299, 13354
get_weekday_by_id() (shinken.daterange.Daterange class method), 13286, 13341
get_weekday_id() (shinken.daterange.Daterange class method), 13286, 13341
got_business_rule (shinken.objects.host.Host attribute), 13312, 13368, 13404
got_business_rule (shinken.objects.service.Service attribute), 13328, 13384, 13420
got_default_realm (shinken.objects.host.Host attribute), 13312, 13368, 13404
got_generation_rule_pattern_change() (in module shinken.util), 13300, 13354
Graph (class in shinken.graph), 13292, 13346
guess_int_or_float() (in module shinken.misc.perfdata), 13304, 13360, 13396
guess_root_problems() (shinken.misc.datamanager.DataManager method), 13304, 13359, 13395

H

handle_close() (shinken.clients.livestatus.LSAsynConnection method), 13301, 13356, 13393
handle_connect() (shinken.clients.livestatus.LSAsynConnection method), 13301, 13356, 13393
handle_read() (shinken.clients.livestatus.LSAsynConnection method), 13301, 13356, 13393
handle_write() (shinken.clients.livestatus.LSAsynConnection method), 13302, 13356, 13393
has() (shinken.basemodule.BaseModule method), 13282, 13337
has() (shinken.objects.itemgroup.Itemgroup method), 13320, 13376, 13413
has() (shinken.objects.timeperiod.Timeperiod method), 13335, 13391, 13427
has_been_checked (shinken.objects.host.Host attribute), 13312, 13368, 13404
has_been_checked (shinken.objects.service.Service attribute), 13328, 13384, 13420
has_plus() (shinken.objects.item.Item method), 13319, 13375, 13411
hash (shinken.objects.host.Host attribute), 13312, 13368, 13404
hash (shinken.objects.service.Service attribute), 13328, 13384, 13420
high_flap_threshold (shinken.objects.host.Host attribute), 13312, 13368, 13404
high_flap_threshold (shinken.objects.service.Service attribute), 13328, 13384, 13420
Host (class in shinken.objects.host), 13309, 13365, 13401
host (shinken.objects.service.Service attribute), 13328, 13384, 13420
host_name (shinken.notification.Notification attribute), 13295, 13350
host_name (shinken.objects.host.Host attribute), 13312, 13368, 13404
host_name (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
host_name (shinken.objects.service.Service attribute), 13328, 13384, 13420
host_name (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13389, 13426
Hostdependencies (class in shinken.objects.hostdependency), 13316, 13372, 13408
Hostdependency (class in shinken.objects.hostdependency), 13316, 13372, 13408
Hostescalation (class in shinken.objects.hostescalation), 13316, 13372, 13408
Hostescalations (class in shinken.objects.hostescalation), 13316, 13372, 13409
HostExtInfo (class in shinken.objects.hostextinfo), 13317, 13373, 13409
Hostgroup (class in shinken.objects.hostgroup), 13317, 13373, 13410
hostgroup_name (shinken.objects.service.Service attribute), 13328, 13384, 13421
Hostgroups (class in shinken.objects.hostgroup), 13318, 13374, 13410
hostgroups (shinken.objects.host.Host attribute), 13312, 13368, 13404
Hosts (class in shinken.objects.host), 13315, 13371, 13407

HostsExtInfo (class in shinken.objects.hostextinfo), 13317, 13373, 13409
hst_srv_sort() (in module shinken.misc.sorter), 13305, 13360, 13396

I

icon_image (shinken.objects.host.Host attribute), 13312, 13368, 13404
icon_image (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
icon_image (shinken.objects.service.Service attribute), 13328, 13384, 13421
icon_image (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13389, 13426
icon_image_alt (shinken.objects.host.Host attribute), 13312, 13368, 13404
icon_image_alt (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
icon_image_alt (shinken.objects.service.Service attribute), 13328, 13384, 13421
icon_image_alt (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13389, 13426
icon_set (shinken.objects.host.Host attribute), 13312, 13368, 13404
icon_set (shinken.objects.service.Service attribute), 13328, 13384, 13421
id (shinken.acknowledge.Acknowledge attribute), 13281, 13336
id (shinken.brok.Brok attribute), 13283, 13338
id (shinken.clients.livestatus.Query attribute), 13302, 13357, 13394
id (shinken.commandcall.CommandCall attribute), 13284, 13339
id (shinken.comment.Comment attribute), 13285, 13340
id (shinken.contactdowntime.ContactDowntime attribute), 13286, 13341
id (shinken.downtime.Downtime attribute), 13289, 13344
id (shinken.objects.businessimpactmodulation.Businessimpactmodulation attribute), 13305, 13361, 13397
id (shinken.objects.checkmodulation.CheckModulation attribute), 13305, 13361, 13397
id (shinken.objects.command.Command attribute), 13306, 13362, 13398
id (shinken.objects.contact.Contact attribute), 13307, 13362, 13399
id (shinken.objects.contactgroup.Contactgroup attribute), 13307, 13363, 13399
id (shinken.objects.discoveryrule.Discoveryrule attribute), 13308, 13364, 13400
id (shinken.objects.discoveryrun.Discoveryrun attribute), 13308, 13364, 13400
id (shinken.objects.escalation.Escalation attribute), 13309, 13365, 13401
id (shinken.objects.host.Host attribute), 13312, 13368, 13404
id (shinken.objects.hostdependency.Hostdependency attribute), 13316, 13372, 13408
id (shinken.objects.hostescalation.Hostescalation attribute), 13316, 13372, 13408
id (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
id (shinken.objects.hostgroup.Hostgroup attribute), 13317, 13374, 13410
id (shinken.objects.itemgroup.Itemgroup attribute), 13320, 13376, 13413
id (shinken.objects.macromodulation.MacroModulation attribute), 13321, 13377, 13413
id (shinken.objects.module.Module attribute), 13321, 13378, 13414
id (shinken.objects.notificationway.NotificationWay attribute), 13322, 13378, 13414
id (shinken.objects.pack.Pack attribute), 13322, 13379, 13415
id (shinken.objects.realm.Realm attribute), 13323, 13380, 13416
id (shinken.objects.resultmodulation.Resultmodulation attribute), 13324, 13380, 13416
id (shinken.objects.service.Service attribute), 13328, 13384, 13421
id (shinken.objects.servicedependency.Servicedependency attribute), 13332, 13389, 13425
id (shinken.objects.serviceescalation.Serviceescalation attribute), 13332, 13389, 13425
id (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13389, 13426
id (shinken.objects.servicegroup.Servicegroup attribute), 13334, 13390, 13426
id (shinken.objects.timeperiod.Timeperiod attribute), 13335, 13391, 13427
id (shinken.objects.trigger.Trigger attribute), 13335, 13392, 13428
id (shinken.worker.Worker attribute), 13300, 13355
impacts (shinken.objects.host.Host attribute), 13312, 13368, 13404
impacts (shinken.objects.service.Service attribute), 13328, 13384, 13421
imported_from (shinken.objects.command.Command attribute), 13306, 13362, 13398
imported_from (shinken.objects.host.Host attribute), 13312, 13368, 13404
imported_from (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
imported_from (shinken.objects.service.Service attribute), 13328, 13384, 13421
imported_from (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13389, 13426

in_checking (shinken.objects.host.Host attribute), 13312, 13368, 13404
in_checking (shinken.objects.service.Service attribute), 13328, 13384, 13421
in_hard_unknown_reach_phase (shinken.objects.host.Host attribute), 13312, 13368, 13404
in_hard_unknown_reach_phase (shinken.objects.service.Service attribute), 13328, 13384, 13421
in_scheduled_downtime (shinken.objects.host.Host attribute), 13312, 13368, 13404
in_scheduled_downtime (shinken.objects.service.Service attribute), 13328, 13385, 13421
in_scheduled_downtime() (shinken.contactdowntime.ContactDowntime method), 13286, 13341
in_scheduled_downtime() (shinken.downtime.Downtime method), 13289, 13344
in_scheduled_downtime_during_last_check (shinken.objects.host.Host attribute), 13312, 13368, 13404
in_scheduled_downtime_during_last_check (shinken.objects.service.Service attribute), 13328, 13385, 13421
INFO (shinken.log.Log attribute), 13292, 13347
info() (shinken.log.Log method), 13293, 13347
init() (shinken.basemodule.BaseModule method), 13282, 13337
init() (shinken.macroresolver.MacroResolver method), 13293, 13348
init_running_properties() (shinken.objects.item.Item method), 13319, 13375, 13411
initial_state (shinken.objects.host.Host attribute), 13312, 13368, 13404
initial_state (shinken.objects.service.Service attribute), 13328, 13385, 13421
inner_class (shinken.objects.businessimpactmodulation.BusinessImpactModulations attribute), 13305, 13361, 13397
inner_class (shinken.objects.checkmodulation.CheckModulations attribute), 13306, 13361, 13398
inner_class (shinken.objects.command.Commands attribute), 13306, 13362, 13398
inner_class (shinken.objects.contact.Contacts attribute), 13307, 13363, 13399
inner_class (shinken.objects.contactgroup.Contactgroups attribute), 13308, 13363, 13400
inner_class (shinken.objects.discoveryrule.Discoveryrules attribute), 13308, 13364, 13400
inner_class (shinken.objects.discoveryrun.Discoveryruns attribute), 13309, 13364, 13401
inner_class (shinken.objects.escalation.Escalations attribute), 13309, 13365, 13401
inner_class (shinken.objects.host.Hosts attribute), 13315, 13371, 13408
inner_class (shinken.objects.hostescalation.Hostescalations attribute), 13316, 13372, 13409
inner_class (shinken.objects.hostextinfo.HostsExtInfo attribute), 13317, 13373, 13409
inner_class (shinken.objects.hostgroup.Hostgroups attribute), 13318, 13374, 13410
inner_class (shinken.objects.macromodulation.MacroModulations attribute), 13321, 13377, 13413
inner_class (shinken.objects.module.Modules attribute), 13321, 13378, 13414
inner_class (shinken.objects.notificationway.NotificationWays attribute), 13322, 13378, 13415
inner_class (shinken.objects.pack.Packs attribute), 13322, 13379, 13415
inner_class (shinken.objects.realm.Realms attribute), 13324, 13380, 13416
inner_class (shinken.objects.resultmodulation.Resultmodulations attribute), 13324, 13380, 13417
inner_class (shinken.objects.service.Services attribute), 13332, 13388, 13424
inner_class (shinken.objects.serviceescalation.Serviceescalations attribute), 13333, 13389, 13425
inner_class (shinken.objects.serviceextinfo.ServicesExtInfo attribute), 13333, 13390, 13426
inner_class (shinken.objects.servicegroup.Servicegroups attribute), 13334, 13390, 13427
inner_class (shinken.objects.timeperiod.Timeperiods attribute), 13335, 13391, 13428
inner_class (shinken.objects.trigger.Triggers attribute), 13335, 13392, 13428
IntegerProp (class in shinken.property), 13296, 13351
internal (shinken.check.Check attribute), 13284, 13339
is_a (shinken.check.Check attribute), 13284, 13339
is_a (shinken.eventhandler.EventHandler attribute), 13291, 13346
is_a (shinken.notification.Notification attribute), 13295, 13350
is_active() (shinken.objects.macromodulation.MacroModulation method), 13321, 13377, 13413
is_administrative() (shinken.notification.Notification method), 13295, 13350
is_alive() (shinken.worker.Worker method), 13300, 13355
is_correct() (shinken.daterange.Daterange method), 13286, 13341
is_correct() (shinken.daterange.MonthWeekDayDaterange method), 13287, 13342
is_correct() (shinken.daterange.StandardDaterange method), 13287, 13342
is_correct() (shinken.objects.checkmodulation.CheckModulation method), 13305, 13361, 13397

is_correct() (shinken.objects.contact.Contact method), 13307, 13362, 13399
is_correct() (shinken.objects.escalation.Escalation method), 13309, 13365, 13401
is_correct() (shinken.objects.host.Host method), 13312, 13368, 13404
is_correct() (shinken.objects.hostextinfo.HostExtInfo method), 13317, 13373, 13409
is_correct() (shinken.objects.item.Item method), 13319, 13375, 13411
is_correct() (shinken.objects.item.Items method), 13320, 13376, 13412
is_correct() (shinken.objects.itemgroup.Itemgroup method), 13320, 13376, 13413
is_correct() (shinken.objects.macromodulation.MacroModulation method), 13321, 13377, 13413
is_correct() (shinken.objects.notificationway.NotificationWay method), 13322, 13378, 13414
is_correct() (shinken.objects.service.Service method), 13328, 13385, 13421
is_correct() (shinken.objects.serviceextinfo.ServiceExtInfo method), 13333, 13389, 13426
is_correct() (shinken.objects.timeperiod.Timeperiod method), 13335, 13391, 13427
is_correct() (shinken.objects.timeperiod.Timeperiods method), 13335, 13391, 13428
is_dependent() (shinken.check.Check method), 13284, 13339
is_eligible() (shinken.objects.escalation.Escalation method), 13309, 13365, 13401
is_escalable() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
is_finished() (shinken.clients.livestatus.LSAsynConnection method), 13302, 13356, 13393
is_finished() (shinken.objects.discoveryrun.Discoveryrun method), 13308, 13364, 13400
is_first_level() (shinken.objects.discoveryrun.Discoveryrun method), 13308, 13364, 13400
is_flapping (shinken.objects.host.Host attribute), 13312, 13368, 13404
is_flapping (shinken.objects.service.Service attribute), 13328, 13385, 13421
is_impact (shinken.objects.host.Host attribute), 13312, 13368, 13404
is_impact (shinken.objects.service.Service attribute), 13328, 13385, 13421
is_killable() (shinken.worker.Worker method), 13300, 13355
is_launchable() (shinken.check.Check method), 13284, 13339
is_launchable() (shinken.eventhandler.EventHandler method), 13291, 13346
is_launchable() (shinken.notification.Notification method), 13295, 13350
is_linked_with_host() (shinken.objects.host.Host method), 13312, 13368, 13404
is_matching() (shinken.objects.matchingitem.MatchingItem method), 13321, 13377, 13414
is_matching_disco_datas() (shinken.objects.matchingitem.MatchingItem method), 13321, 13377, 13414
is_max_attempts() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13417
is_mortal() (shinken.worker.Worker method), 13300, 13355
is_no_action_dependent() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13418
is_no_check_dependent() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13418
is_problem (shinken.objects.host.Host attribute), 13312, 13368, 13404
is_problem (shinken.objects.service.Service attribute), 13328, 13385, 13421
is_state() (shinken.objects.host.Host method), 13312, 13368, 13404
is_state() (shinken.objects.service.Service method), 13328, 13385, 13421
is_time_day_invalid() (shinken.daterange.Daterange method), 13286, 13341
is_time_day_valid() (shinken.daterange.Daterange method), 13286, 13341
is_time_valid() (shinken.daterange.Daterange method), 13286, 13341
is_time_valid() (shinken.daterange.Timerange method), 13287, 13342
is_time_valid() (shinken.objects.timeperiod.Timeperiod method), 13335, 13391, 13427
is_tpl() (shinken.objects.item.Item method), 13319, 13375, 13411
is_tty() (in module shinken.log), 13293, 13348
is_valid() (shinken.commandcall.CommandCall method), 13285, 13339
is_valid() (shinken.complexexpression.ComplexExpressionNode method), 13285, 13340
is_valid() (shinken.dependencynode.DependencyNode method), 13288, 13343
is_volatile (shinken.objects.service.Service attribute), 13328, 13385, 13421
Item (class in shinken.objects.item), 13318, 13374, 13410
Itemgroup (class in shinken.objects.itemgroup), 13320, 13376, 13413
Itemgroups (class in shinken.objects.itemgroup), 13320, 13377, 13413
Items (class in shinken.objects.item), 13319, 13375, 13411

items() (shinken.sorteddict.SortedDict method), 13298, 13353
iteritems() (shinken.sorteddict.SortedDict method), 13298, 13353
iterkeys() (shinken.sorteddict.SortedDict method), 13298, 13353
itervalues() (shinken.sorteddict.SortedDict method), 13298, 13353

J

join() (shinken.worker.Worker method), 13300, 13355

K

keys() (shinken.sorteddict.SortedDict method), 13298, 13353
kill__() (shinken.action.Action method), 13281, 13336

L

largest_item() (shinken.sorteddict.SortedDict method), 13298, 13353
largest_key() (shinken.sorteddict.SortedDict method), 13298, 13353
last_chk (shinken.objects.host.Host attribute), 13312, 13368, 13404
last_chk (shinken.objects.service.Service attribute), 13328, 13385, 13421
last_event_id (shinken.objects.host.Host attribute), 13312, 13368, 13404
last_event_id (shinken.objects.service.Service attribute), 13328, 13385, 13421
last_hard_state (shinken.objects.host.Host attribute), 13312, 13368, 13405
last_hard_state (shinken.objects.service.Service attribute), 13328, 13385, 13421
last_hard_state_change (shinken.objects.host.Host attribute), 13312, 13368, 13405
last_hard_state_change (shinken.objects.service.Service attribute), 13328, 13385, 13421
last_hard_state_id (shinken.objects.host.Host attribute), 13312, 13368, 13405
last_hard_state_id (shinken.objects.service.Service attribute), 13328, 13385, 13421
last_notification (shinken.objects.host.Host attribute), 13312, 13368, 13405
last_notification (shinken.objects.service.Service attribute), 13328, 13385, 13421
last_perf_data (shinken.objects.host.Host attribute), 13312, 13368, 13405
last_perf_data (shinken.objects.service.Service attribute), 13329, 13385, 13421
last_problem_id (shinken.objects.host.Host attribute), 13313, 13368, 13405
last_problem_id (shinken.objects.service.Service attribute), 13329, 13385, 13421
last_state (shinken.objects.host.Host attribute), 13313, 13368, 13405
last_state (shinken.objects.service.Service attribute), 13329, 13385, 13421
last_state_change (shinken.objects.host.Host attribute), 13313, 13368, 13405
last_state_change (shinken.objects.service.Service attribute), 13329, 13385, 13421
last_state_change_earlier() (in module shinken.misc.sorter), 13305, 13360, 13396
last_state_id (shinken.objects.host.Host attribute), 13313, 13368, 13405
last_state_id (shinken.objects.service.Service attribute), 13329, 13385, 13421
last_state_type (shinken.objects.host.Host attribute), 13313, 13369, 13405
last_state_type (shinken.objects.service.Service attribute), 13329, 13385, 13421
last_state_update (shinken.objects.host.Host attribute), 13313, 13369, 13405
last_state_update (shinken.objects.service.Service attribute), 13329, 13385, 13421
last_time_critical (shinken.objects.service.Service attribute), 13329, 13385, 13421
last_time_down (shinken.objects.host.Host attribute), 13313, 13369, 13405
last_time_non_ok_or_up() (shinken.objects.host.Host method), 13313, 13369, 13405
last_time_non_ok_or_up() (shinken.objects.service.Service method), 13329, 13385, 13421
last_time_ok (shinken.objects.service.Service attribute), 13329, 13385, 13421
last_time_unknown (shinken.objects.service.Service attribute), 13329, 13385, 13421
last_time_unreachable (shinken.objects.host.Host attribute), 13313, 13369, 13405
last_time_up (shinken.objects.host.Host attribute), 13313, 13369, 13405
last_time_warning (shinken.objects.service.Service attribute), 13329, 13385, 13421
LATE_CONFIGURATION (shinken.basemodule.ModulePhases attribute), 13283, 13338
late_linkify_c_by_commands() (shinken.objects.contact.Contacts method), 13307, 13363, 13399
late_linkify_cw_by_commands() (shinken.objects.checkmodulation.CheckModulation method), 13306, 13361, 13397
late_linkify_h_by_commands() (shinken.objects.host.Hosts method), 13315, 13371, 13408

late_linkify_nw_by_commands() (shinken.objects.notificationway.NotificationWay method), 13322, 13378, 13414
late_linkify_s_by_commands() (shinken.objects.service.Services method), 13332, 13388, 13424
late_linkify_with_command() (shinken.commandcall.CommandCall method), 13285, 13339
late_relink_done (shinken.commandcall.CommandCall attribute), 13285, 13340
latency (shinken.objects.host.Host attribute), 13313, 13369, 13405
latency (shinken.objects.service.Service attribute), 13329, 13385, 13422
launch() (shinken.objects.discoveryrun.Discoveryrun method), 13308, 13364, 13401
launch_check() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13418
launch_new_checks() (shinken.worker.Worker method), 13300, 13355
launch_query() (shinken.clients.livestatus.LSSyncConnection method), 13302, 13357, 13393
launch_raw_query() (shinken.clients.livestatus.LSAsynConnection method), 13302, 13356, 13393
launch_raw_query() (shinken.clients.livestatus.LSConnection method), 13302, 13357, 13393
linkify() (shinken.objects.businessimpactmodulation.BusinessImpactModulations method), 13305, 13361, 13397
linkify() (shinken.objects.checkmodulation.CheckModulations method), 13306, 13361, 13398
linkify() (shinken.objects.contact.Contacts method), 13307, 13363, 13399
linkify() (shinken.objects.contactgroup.Contactgroups method), 13308, 13364, 13400
linkify() (shinken.objects.discoveryrun.Discoveryruns method), 13309, 13364, 13401
linkify() (shinken.objects.escalation.Escalations method), 13309, 13365, 13401
linkify() (shinken.objects.host.Hosts method), 13316, 13371, 13408
linkify() (shinken.objects.hostdependency.Hostdependencies method), 13316, 13372, 13408
linkify() (shinken.objects.hostgroup.Hostgroups method), 13318, 13374, 13410
linkify() (shinken.objects.macromodulation.MacroModulations method), 13321, 13377, 13413
linkify() (shinken.objects.module.Modules method), 13322, 13378, 13414
linkify() (shinken.objects.notificationway.NotificationWays method), 13322, 13378, 13415
linkify() (shinken.objects.realm.Realms method), 13324, 13380, 13416
linkify() (shinken.objects.resultmodulation.Resultmodulations method), 13324, 13380, 13417
linkify() (shinken.objects.service.Services method), 13332, 13388, 13424
linkify() (shinken.objects.servicedependency.Servicedependencies method), 13332, 13388, 13425
linkify() (shinken.objects.servicegroup.Servicegroups method), 13334, 13390, 13427
linkify() (shinken.objects.timeperiod.Timeperiod method), 13335, 13391, 13427
linkify() (shinken.objects.timeperiod.Timeperiods method), 13335, 13391, 13428
linkify_cg_by_cont() (shinken.objects.contactgroup.Contactgroups method), 13308, 13364, 13400
linkify_cm_by_tp() (shinken.objects.businessimpactmodulation.Businessim method), 13305, 13361, 13397
linkify_command_list_with_commands() (shinken.objects.item.Items method), 13320, 13376, 13412
linkify_es_by_h() (shinken.objects.escalation.Escalations method), 13309, 13365, 13401
linkify_es_by_s() (shinken.objects.escalation.Escalations method), 13309, 13365, 13401
linkify_h_by_h() (shinken.objects.host.Hosts method), 13316, 13371, 13408
linkify_h_by_hd() (shinken.objects.hostdependency.Hostdependencies method), 13316, 13372, 13408
linkify_h_by_hg() (shinken.objects.host.Hosts method), 13316, 13372, 13408
linkify_h_by_realms() (shinken.objects.host.Hosts method), 13316, 13372, 13408
linkify_hd_by_h() (shinken.objects.hostdependency.Hostdependencies method), 13316, 13372, 13408
linkify_hd_by_tp() (shinken.objects.hostdependency.Hostdependencies method), 13316, 13372, 13408
linkify_hg_by_hst() (shinken.objects.hostgroup.Hostgroups method), 13318, 13374, 13410
linkify_hg_by_realms() (shinken.objects.hostgroup.Hostgroups method), 13318, 13374, 13410
linkify_one_command_with_commands() (shinken.objects.item.Item method), 13319, 13375, 13411
linkify_one_command_with_commands() (shinken.objects.item.Items method), 13320, 13376, 13412
linkify_p_by_p() (shinken.objects.realm.Realms method), 13324, 13380, 13416
linkify_rm_by_tp() (shinken.objects.resultmodulation.Resultmodulations method), 13324, 13380, 13417
linkify_s_by_hst() (shinken.objects.service.Services method), 13332, 13388, 13424
linkify_s_by_plug() (shinken.objects.module.Modules method), 13322, 13378, 13414

linkify_s_by_sd() (shinken.objects.servicedependency.Servicedependencies method), 13332, 13388, 13425
linkify_s_by_sg() (shinken.objects.service.Services method), 13332, 13388, 13424
linkify_sd_by_s() (shinken.objects.servicedependency.Servicedependencies method), 13332, 13388, 13425
linkify_sd_by_tp() (shinken.objects.servicedependency.Servicedependencies method), 13332, 13389, 13425
linkify_sg_by_srv() (shinken.objects.servicegroup.Servicegroups method), 13334, 13390, 13427
linkify_templates() (shinken.objects.item.Items method), 13320, 13376, 13412
linkify_with_business_impact_modulations() (shinken.objects.item.Items method), 13320, 13376, 13412
linkify_with_checkmodulations() (shinken.objects.item.Items method), 13320, 13376, 13412
linkify_with_contacts() (shinken.objects.item.Items method), 13320, 13376, 13412
linkify_with_escalations() (shinken.objects.item.Items method), 13320, 13376, 13412
linkify_with_macromodulations() (shinken.objects.item.Items method), 13320, 13376, 13412
linkify_with_notificationways() (shinken.objects.contact.Contacts method), 13307, 13363, 13399
linkify_with_resultmodulations() (shinken.objects.item.Items method), 13320, 13376, 13412
linkify_with_timeperiods() (shinken.objects.item.Items method), 13320, 13376, 13412
linkify_with_triggers() (shinken.objects.item.Item method), 13319, 13375, 13411
linkify_with_triggers() (shinken.objects.item.Items method), 13320, 13376, 13412
list_all_elements() (shinken.dependencynode.DependencyNode method), 13288, 13343
ListProp (class in shinken.property), 13297, 13352
Load (class in shinken.load), 13292, 13347
load() (shinken.misc.datamanager.DataManager method), 13304, 13359, 13395
load() (shinken.modulesmanager.ModulesManager method), 13294, 13349
load_and_init() (shinken.modulesmanager.ModulesManager method), 13294, 13349
load_file() (shinken.objects.pack.Packs method), 13323, 13379, 13415
load_file() (shinken.objects.trigger.Triggers method), 13335, 13392, 13428
load_global_conf() (shinken.objects.item.Item class method), 13319, 13375, 13411
load_modules() (shinken.modulesmanager.ModulesManager method), 13294, 13349
load_obj() (shinken.log.Log method), 13293, 13347
load_objects() (shinken.objects.trigger.Triggers method), 13335, 13392, 13428
Log (class in shinken.log), 13292, 13347
log() (shinken.log.Log method), 13293, 13347
long_output (shinken.check.Check attribute), 13284, 13339
long_output (shinken.eventhandler.EventHandler attribute), 13291, 13346
long_output (shinken.objects.host.Host attribute), 13313, 13369, 13405
long_output (shinken.objects.service.Service attribute), 13329, 13385, 13422
look_for_timeout() (shinken.clients.livestatus.LSAsynConnection method), 13302, 13356, 13393
loop_check() (shinken.graph.Graph method), 13292, 13347
low_flap_threshold (shinken.objects.host.Host attribute), 13313, 13369, 13405
low_flap_threshold (shinken.objects.service.Service attribute), 13329, 13385, 13422
LSAsynConnection (class in shinken.clients.livestatus), 13301, 13356, 13393
LSConnectionPool (class in shinken.clients.livestatus), 13302, 13357, 13393
LSSyncConnection (class in shinken.clients.livestatus), 13302, 13357, 13393

M

MacroModulation (class in shinken.objects.macromodulation), 13321, 13377, 13413
MacroModulations (class in shinken.objects.macromodulation), 13321, 13377, 13413
macromodulations (shinken.objects.host.Host attribute), 13313, 13369, 13405
macromodulations (shinken.objects.service.Service attribute), 13329, 13385, 13422
MacroResolver (class in shinken.macroresolver), 13293, 13348
macros (shinken.macroresolver.MacroResolver attribute), 13293, 13348
macros (shinken.notification.Notification attribute), 13295, 13350
macros (shinken.objects.checkmodulation.CheckModulation attribute), 13306, 13361, 13397
macros (shinken.objects.contact.Contact attribute), 13307, 13362, 13399
macros (shinken.objects.contactgroup.Contactgroup attribute), 13308, 13363, 13399

macros (shinken.objects.discoveryrule.Discoveryrule attribute), 13308, 13364, 13400
macros (shinken.objects.host.Host attribute), 13313, 13369, 13405
macros (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
macros (shinken.objects.hostgroup.Hostgroup attribute), 13317, 13374, 13410
macros (shinken.objects.item.Item attribute), 13319, 13375, 13411
macros (shinken.objects.macromodulation.MacroModulation attribute), 13321, 13377, 13413
macros (shinken.objects.module.Module attribute), 13321, 13378, 13414
macros (shinken.objects.notificationway.NotificationWay attribute), 13322, 13378, 13414
macros (shinken.objects.realm.Realm attribute), 13323, 13380, 13416
macros (shinken.objects.service.Service attribute), 13329, 13385, 13422
macros (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13389, 13426
macros (shinken.objects.servicegroup.Servicegroup attribute), 13334, 13390, 13426
main() (in module shinken.clients.LSB), 13301, 13356, 13393
main() (shinken.basemodule.BaseModule method), 13282, 13337
maintenance_period (shinken.objects.host.Host attribute), 13313, 13369, 13405
maintenance_period (shinken.objects.service.Service attribute), 13329, 13385, 13422
manage_brok() (shinken.basemodule.BaseModule method), 13282, 13337
manage_finished_checks() (shinken.worker.Worker method), 13300, 13355
manage_internal_check() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13418
manage_signal() (shinken.basemodule.BaseModule method), 13282, 13337
manage_stalking() (shinken.objects.host.Host method), 13313, 13369, 13405
manage_stalking() (shinken.objects.service.Service method), 13329, 13385, 13422
MatchingItem (class in shinken.objects.matchingitem), 13321, 13377, 13414
max_check_attempts (shinken.objects.host.Host attribute), 13313, 13369, 13405
max_check_attempts (shinken.objects.service.Service attribute), 13329, 13385, 13422
md5crypt() (in module shinken.misc.md5crypt), 13304, 13360, 13396
mean() (in module shinken.clients.LSB), 13301, 13356, 13393
median() (in module shinken.clients.LSB), 13301, 13356, 13393
memoized (class in shinken.memoized), 13293, 13348
merge() (shinken.objects.hostextinfo.HostsExtInfo method), 13317, 13373, 13409
merge() (shinken.objects.serviceextinfo.ServicesExtInfo method), 13333, 13390, 13426
merge_extinfo() (shinken.objects.hostextinfo.HostsExtInfo method), 13317, 13373, 13409
merge_extinfo() (shinken.objects.serviceextinfo.ServicesExtInfo method), 13333, 13390, 13426
Message (class in shinken.message), 13294, 13348
Metric (class in shinken.misc.perfdata), 13304, 13360, 13396
modified_attributes (shinken.objects.host.Host attribute), 13313, 13369, 13405
modified_attributes (shinken.objects.service.Service attribute), 13329, 13385, 13422
Module (class in shinken.objects.module), 13321, 13377, 13414
module_return() (shinken.objects.resultmodulation.Resultmodulation method), 13324, 13380, 13416
module_type (shinken.check.Check attribute), 13284, 13339
module_type (shinken.commandcall.CommandCall attribute), 13285, 13340
module_type (shinken.eventhandler.EventHandler attribute), 13291, 13346
module_type (shinken.notification.Notification attribute), 13295, 13350
module_type (shinken.objects.command.Command attribute), 13306, 13362, 13398
ModulePhases (class in shinken.basemodule), 13282, 13337
Modules (class in shinken.objects.module), 13321, 13378, 13414
ModulesManager (class in shinken.modulesmanager), 13294, 13349
MonthDateDaterange (class in shinken.daterange), 13286, 13341
MonthDayDaterange (class in shinken.daterange), 13287, 13342
months (shinken.daterange.Daterange attribute), 13286, 13341
MonthWeekDayDaterange (class in shinken.daterange), 13287, 13342
my_own_business_impact (shinken.objects.host.Host attribute), 13313, 13369, 13405
my_own_business_impact (shinken.objects.service.Service attribute), 13329, 13386, 13422
my_type (shinken.brok.Brok attribute), 13283, 13338

my_type (shinken.check.Check attribute), 13284, 13339
my_type (shinken.commandcall.CommandCall attribute), 13285, 13340
my_type (shinken.eventhandler.EventHandler attribute), 13291, 13346
my_type (shinken.macroresolver.MacroResolver attribute), 13293, 13348
my_type (shinken.message.Message attribute), 13294, 13348
my_type (shinken.notification.Notification attribute), 13295, 13350
my_type (shinken.objects.businessimpactmodulation.Businessimpactmodulation attribute), 13305, 13361, 13397
my_type (shinken.objects.checkmodulation.CheckModulation attribute), 13306, 13361, 13397
my_type (shinken.objects.command.Command attribute), 13306, 13362, 13398
my_type (shinken.objects.contact.Contact attribute), 13307, 13362, 13399
my_type (shinken.objects.contactgroup.Contactgroup attribute), 13308, 13363, 13400
my_type (shinken.objects.discoveryrule.Discoveryrule attribute), 13308, 13364, 13400
my_type (shinken.objects.discoveryrun.Discoveryrun attribute), 13309, 13364, 13401
my_type (shinken.objects.escalation.Escalation attribute), 13309, 13365, 13401
my_type (shinken.objects.host.Host attribute), 13313, 13369, 13405
my_type (shinken.objects.hostdependency.Hostdependency attribute), 13316, 13372, 13408
my_type (shinken.objects.hostescalation.Hostescalation attribute), 13316, 13372, 13408
my_type (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
my_type (shinken.objects.hostgroup.Hostgroup attribute), 13318, 13374, 13410
my_type (shinken.objects.macromodulation.MacroModulation attribute), 13321, 13377, 13413
my_type (shinken.objects.module.Module attribute), 13321, 13378, 13414
my_type (shinken.objects.notificationway.NotificationWay attribute), 13322, 13378, 13414
my_type (shinken.objects.pack.Pack attribute), 13322, 13379, 13415
my_type (shinken.objects.realm.Realm attribute), 13323, 13380, 13416
my_type (shinken.objects.resultmodulation.Resultmodulation attribute), 13324, 13380, 13416
my_type (shinken.objects.service.Service attribute), 13329, 13386, 13422
my_type (shinken.objects.servicedependency.Servicedependency attribute), 13332, 13389, 13425
my_type (shinken.objects.serviceescalation.Serviceescalation attribute), 13332, 13389, 13425
my_type (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13390, 13426
my_type (shinken.objects.servicegroup.Servicegroup attribute), 13334, 13390, 13426
my_type (shinken.objects.timeperiod.Timeperiod attribute), 13335, 13391, 13427
my_type (shinken.objects.trigger.Trigger attribute), 13335, 13392, 13428
myip() (in module shinken.easter), 13290, 13345

N

naheulbeuk() (in module shinken.easter), 13290, 13345
name (shinken.objects.command.Command attribute), 13306, 13362, 13398
name (shinken.objects.host.Host attribute), 13313, 13369, 13405
name (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
name (shinken.objects.service.Service attribute), 13329, 13386, 13422
name (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13390, 13426
name_property (shinken.objects.businessimpactmodulation.Businessimpact attribute), 13305, 13361, 13397
name_property (shinken.objects.checkmodulation.CheckModulations attribute), 13306, 13361, 13398
name_property (shinken.objects.command.Commands attribute), 13306, 13362, 13398
name_property (shinken.objects.contact.Contacts attribute), 13307, 13363, 13399
name_property (shinken.objects.contactgroup.Contactgroups attribute), 13308, 13364, 13400
name_property (shinken.objects.discoveryrule.Discoveryrules attribute), 13308, 13364, 13400
name_property (shinken.objects.discoveryrun.Discoveryruns attribute), 13309, 13365, 13401
name_property (shinken.objects.escalation.Escalations attribute), 13309, 13365, 13401
name_property (shinken.objects.host.Hosts attribute), 13316, 13372, 13408
name_property (shinken.objects.hostescalation.Hostescalations attribute), 13316, 13372, 13409
name_property (shinken.objects.hostextinfo.HostsExtInfo attribute), 13317, 13373, 13409
name_property (shinken.objects.hostgroup.Hostgroups attribute), 13318, 13374, 13410
name_property (shinken.objects.macromodulation.MacroModulations attribute), 13321, 13377, 13413
name_property (shinken.objects.module.Modules attribute), 13322, 13378, 13414
name_property (shinken.objects.notificationway.NotificationWays attribute), 13322, 13378, 13415

name_property (shinken.objects.pack.Packs attribute), 13323, 13379, 13415
name_property (shinken.objects.realm.Realms attribute), 13324, 13380, 13416
name_property (shinken.objects.resultmodulation.Resultmodulations attribute), 13324, 13380, 13417
name_property (shinken.objects.serviceescalation.Serviceescalations attribute), 13333, 13389, 13425
name_property (shinken.objects.serviceextinfo.ServicesExtInfo attribute), 13333, 13390, 13426
name_property (shinken.objects.servicegroup.Servicegroups attribute), 13334, 13390, 13427
name_property (shinken.objects.timeperiod.Timeperiods attribute), 13335, 13391, 13428
name_property (shinken.objects.trigger.Triggers attribute), 13335, 13392, 13428
new_inner_member() (shinken.objects.checkmodulation.CheckModulations method), 13306, 13361, 13398
new_inner_member() (shinken.objects.notificationway.NotificationWays method), 13322, 13378, 13415
next_chk (shinken.objects.host.Host attribute), 13313, 13369, 13405
next_chk (shinken.objects.service.Service attribute), 13329, 13386, 13422
nighty_five_percent() (in module shinken.util), 13300, 13354
no_block_read() (in module shinken.action), 13281, 13336
no_loop_in_parents() (shinken.objects.host.Hosts method), 13316, 13372, 13408
no_more_a_problem() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13418
notes (shinken.objects.host.Host attribute), 13313, 13369, 13405
notes (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
notes (shinken.objects.service.Service attribute), 13329, 13386, 13422
notes (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13390, 13426
notes_url (shinken.objects.host.Host attribute), 13313, 13369, 13405
notes_url (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
notes_url (shinken.objects.service.Service attribute), 13329, 13386, 13422
notes_url (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13390, 13426
notif_nb (shinken.notification.Notification attribute), 13295, 13350
Notification (class in shinken.notification), 13295, 13349
notification_interval (shinken.objects.host.Host attribute), 13313, 13369, 13405
notification_interval (shinken.objects.service.Service attribute), 13329, 13386, 13422
notification_is_blocked_by_contact() (shinken.objects.host.Host method), 13313, 13369, 13405
notification_is_blocked_by_contact() (shinken.objects.service.Service method), 13329, 13386, 13422
notification_is_blocked_by_item() (shinken.objects.host.Host method), 13313, 13369, 13405
notification_is_blocked_by_item() (shinken.objects.service.Service method), 13329, 13386, 13422
notification_options (shinken.objects.host.Host attribute), 13313, 13369, 13405
notification_options (shinken.objects.service.Service attribute), 13329, 13386, 13422
notification_period (shinken.objects.host.Host attribute), 13313, 13369, 13405
notification_period (shinken.objects.service.Service attribute), 13329, 13386, 13422
notification_type (shinken.notification.Notification attribute), 13296, 13350
notifications_enabled (shinken.objects.host.Host attribute), 13313, 13369, 13405
notifications_enabled (shinken.objects.service.Service attribute), 13329, 13386, 13422
notifications_in_progress (shinken.objects.host.Host attribute), 13313, 13369, 13406
notifications_in_progress (shinken.objects.service.Service attribute), 13329, 13386, 13422
NotificationWay (class in shinken.objects.notificationway), 13322, 13378, 13414
NotificationWays (class in shinken.objects.notificationway), 13322, 13378, 13415
notified_contacts (shinken.objects.host.Host attribute), 13313, 13369, 13406
notified_contacts (shinken.objects.service.Service attribute), 13329, 13386, 13422
NOTSET (shinken.log.Log attribute), 13292, 13347

O

obsess_over_host (shinken.objects.host.Host attribute), 13313, 13369, 13406
obsess_over_service (shinken.objects.service.Service attribute), 13329, 13386, 13422
ok_up (shinken.objects.host.Host attribute), 13313, 13369, 13406
ok_up (shinken.objects.service.Service attribute), 13330, 13386, 13422

old_properties (shinken.objects.contact.Contact attribute), 13307, 13363, 13399
old_properties (shinken.objects.host.Host attribute), 13313, 13369, 13406
old_properties (shinken.objects.notificationway.NotificationWay attribute), 13322, 13378, 13414
old_properties (shinken.objects.service.Service attribute), 13330, 13386, 13422
old_properties_names_to_new() (shinken.objects.item.Item method), 13319, 13375, 13411
old_properties_names_to_new() (shinken.objects.item.Items method), 13320, 13376, 13412
only_related_to() (in module shinken.misc.filter), 13304, 13359, 13395
optimize_service_search() (shinken.objects.service.Services method), 13332, 13388, 13424
output (shinken.check.Check attribute), 13284, 13339
output (shinken.eventhandler.EventHandler attribute), 13291, 13346
output (shinken.notification.Notification attribute), 13296, 13350
output (shinken.objects.host.Host attribute), 13314, 13369, 13406
output (shinken.objects.service.Service attribute), 13330, 13386, 13422

P

Pack (class in shinken.objects.pack), 13322, 13379, 13415
pack_id (shinken.objects.host.Host attribute), 13314, 13369, 13406
Packs (class in shinken.objects.pack), 13322, 13379, 13415
parallelize_check (shinken.objects.service.Service attribute), 13330, 13386, 13422
parent_dependencies (shinken.objects.host.Host attribute), 13314, 13369, 13406
parent_dependencies (shinken.objects.service.Service attribute), 13330, 13386, 13422
parents (shinken.objects.host.Host attribute), 13314, 13369, 13406
passive_checks_enabled (shinken.objects.host.Host attribute), 13314, 13370, 13406
passive_checks_enabled (shinken.objects.service.Service attribute), 13330, 13386, 13422
pending_flex_downtime (shinken.objects.host.Host attribute), 13314, 13370, 13406
pending_flex_downtime (shinken.objects.service.Service attribute), 13330, 13386, 13422
percent_state_change (shinken.objects.host.Host attribute), 13314, 13370, 13406
percent_state_change (shinken.objects.service.Service attribute), 13330, 13386, 13422
perdu() (in module shinken.easter), 13290, 13345
perf_data (shinken.check.Check attribute), 13284, 13339
perf_data (shinken.eventhandler.EventHandler attribute), 13291, 13346
perf_data (shinken.objects.host.Host attribute), 13314, 13370, 13406
perf_data (shinken.objects.service.Service attribute), 13330, 13386, 13422
PerfDatas (class in shinken.misc.perfdata), 13304, 13360, 13396
poller_tag (shinken.check.Check attribute), 13284, 13339
poller_tag (shinken.commandcall.CommandCall attribute), 13285, 13340
poller_tag (shinken.objects.command.Command attribute), 13306, 13362, 13398
poller_tag (shinken.objects.host.Host attribute), 13314, 13370, 13406
poller_tag (shinken.objects.service.Service attribute), 13330, 13386, 13422
pop() (shinken.sorteddict.SortedDict method), 13298, 13353
popitem() (shinken.sorteddict.SortedDict method), 13299, 13353
prepare() (shinken.brok.Brok method), 13283, 13338
prepare_for_conf_sending() (shinken.objects.item.Item method), 13319, 13375, 13411
prepare_for_satellites_conf() (shinken.objects.realm.Realm method), 13323, 13380, 13416
prepare_for_satellites_conf() (shinken.objects.realm.Realms method), 13324, 13380, 13416
prepare_for_sending() (shinken.objects.item.Items method), 13320, 13376, 13412
prepare_notification_for_sending() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13418
print_date() (in module shinken.util), 13300, 13354
problem_has_been_acknowledged (shinken.objects.host.Host attribute), 13314, 13370, 13406
problem_has_been_acknowledged (shinken.objects.service.Service attribute), 13330, 13386, 13422
process_perf_data (shinken.objects.host.Host attribute), 13314, 13370, 13406
process_perf_data (shinken.objects.service.Service attribute), 13330, 13386, 13422
properties (shinken.acknowledge.Acknowledge attribute), 13281, 13336
properties (shinken.check.Check attribute), 13284, 13339

properties (shinken.commandcall.CommandCall attribute), 13285, 13340
properties (shinken.comment.Comment attribute), 13285, 13340
properties (shinken.contactdowntime.ContactDowntime attribute), 13286, 13341
properties (shinken.downtime.Downtime attribute), 13289, 13344
properties (shinken.eventhandler.EventHandler attribute), 13291, 13346
properties (shinken.notification.Notification attribute), 13296, 13350
properties (shinken.objects.businessimpactmodulation.BusinessImpactModulation attribute), 13305, 13361, 13397
properties (shinken.objects.checkmodulation.CheckModulation attribute), 13306, 13361, 13397
properties (shinken.objects.command.Command attribute), 13306, 13362, 13398
properties (shinken.objects.contact.Contact attribute), 13307, 13363, 13399
properties (shinken.objects.contactgroup.Contactgroup attribute), 13308, 13363, 13400
properties (shinken.objects.discoveryrule.Discoveryrule attribute), 13308, 13364, 13400
properties (shinken.objects.discoveryrun.Discoveryrun attribute), 13309, 13364, 13401
properties (shinken.objects.escalation.Escalation attribute), 13309, 13365, 13401
properties (shinken.objects.host.Host attribute), 13314, 13370, 13406
properties (shinken.objects.hostdependency.Hostdependency attribute), 13316, 13372, 13408
properties (shinken.objects.hostescalation.Hostescalation attribute), 13316, 13372, 13409
properties (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
properties (shinken.objects.hostgroup.Hostgroup attribute), 13318, 13374, 13410
properties (shinken.objects.item.Item attribute), 13319, 13375, 13411
properties (shinken.objects.itemgroup.Itemgroup attribute), 13320, 13376, 13413
properties (shinken.objects.macromodulation.MacroModulation attribute), 13321, 13377, 13413
properties (shinken.objects.module.Module attribute), 13321, 13378, 13414
properties (shinken.objects.notificationway.NotificationWay attribute), 13322, 13378, 13414
properties (shinken.objects.pack.Pack attribute), 13322, 13379, 13415
properties (shinken.objects.realm.Realm attribute), 13323, 13380, 13416
properties (shinken.objects.resultmodulation.Resultmodulation attribute), 13324, 13380, 13416
properties (shinken.objects.service.Service attribute), 13330, 13386, 13423
properties (shinken.objects.servicedependency.Servicedependency attribute), 13332, 13389, 13425
properties (shinken.objects.serviceescalation.Serviceescalation attribute), 13332, 13389, 13425
properties (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13390, 13426
properties (shinken.objects.servicegroup.Servicegroup attribute), 13334, 13390, 13426
properties (shinken.objects.timeperiod.Timeperiod attribute), 13335, 13391, 13427
properties (shinken.objects.trigger.Trigger attribute), 13335, 13392, 13428
put() (shinken.clients.livestatus.Query method), 13302, 13357, 13394
pythonize() (shinken.objects.command.Command method), 13306, 13362, 13398
pythonize() (shinken.objects.item.Item method), 13319, 13375, 13411
pythonize() (shinken.objects.item.Items method), 13320, 13376, 13412
pythonize() (shinken.objects.itemgroup.Itemgroup method), 13320, 13377, 13413
pythonize() (shinken.objects.realm.Realm method), 13323, 13380, 13416
pythonize() (shinken.objects.resultmodulation.Resultmodulation method), 13324, 13380, 13417
pythonize() (shinken.property.BoolProp method), 13296, 13351
pythonize() (shinken.property.CharProp method), 13297, 13352
pythonize() (shinken.property.FloatProp method), 13297, 13351
pythonize() (shinken.property.IntegerProp method), 13297, 13351
pythonize() (shinken.property.ListProp method), 13297, 13352
pythonize() (shinken.property.StringProp method), 13297, 13352

Q

Query (class in shinken.clients.livestatus), 13302, 13357, 13394
QueryGenerator (class in shinken.clients.LSB), 13301, 13356, 13392
quit() (shinken.log.Log method), 13293, 13348

R

raise_alert_log_entry() (shinken.objects.host.Host method), 13314, 13370, 13406
raise_alert_log_entry() (shinken.objects.service.Service method), 13330, 13386, 13423

raise_cancel_downtime_log_entry() (shinken.objects.contact.Contact method), 13307, 13363, 13399
raise_cancel_downtime_log_entry() (shinken.objects.host.Host method), 13314, 13370, 13406
raise_cancel_downtime_log_entry() (shinken.objects.service.Service method), 13330, 13386, 13423
raise_dependencies_check() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13418
raise_enter_downtime_log_entry() (shinken.objects.contact.Contact method), 13307, 13363, 13399
raise_enter_downtime_log_entry() (shinken.objects.host.Host method), 13314, 13370, 13406
raise_enter_downtime_log_entry() (shinken.objects.service.Service method), 13330, 13386, 13423
raise_event_handler_log_entry() (shinken.objects.host.Host method), 13314, 13370, 13406
raise_event_handler_log_entry() (shinken.objects.service.Service method), 13330, 13386, 13423
raise_exit_downtime_log_entry() (shinken.objects.contact.Contact method), 13307, 13363, 13399
raise_exit_downtime_log_entry() (shinken.objects.host.Host method), 13314, 13370, 13406
raise_exit_downtime_log_entry() (shinken.objects.service.Service method), 13330, 13386, 13423
raise_flapping_start_log_entry() (shinken.objects.host.Host method), 13314, 13370, 13406
raise_flapping_start_log_entry() (shinken.objects.service.Service method), 13330, 13386, 13423
raise_flapping_stop_log_entry() (shinken.objects.host.Host method), 13314, 13370, 13406
raise_flapping_stop_log_entry() (shinken.objects.service.Service method), 13330, 13386, 13423
raise_freshness_log_entry() (shinken.objects.host.Host method), 13314, 13370, 13406
raise_freshness_log_entry() (shinken.objects.service.Service method), 13330, 13386, 13423
raise_initial_state() (shinken.objects.host.Host method), 13314, 13370, 13406
raise_initial_state() (shinken.objects.service.Service method), 13330, 13387, 13423
raise_no_next_check_log_entry() (shinken.objects.host.Host method), 13314, 13370, 13406
raise_no_next_check_log_entry() (shinken.objects.service.Service method), 13330, 13387, 13423
raise_notification_log_entry() (shinken.objects.host.Host method), 13314, 13370, 13406
raise_notification_log_entry() (shinken.objects.service.Service method), 13330, 13387, 13423
reactionner_tag (shinken.check.Check attribute), 13284, 13339
reactionner_tag (shinken.commandcall.CommandCall attribute), 13285, 13340
reactionner_tag (shinken.eventhandler.EventHandler attribute), 13291, 13346
reactionner_tag (shinken.notification.Notification attribute), 13296, 13350
reactionner_tag (shinken.objects.command.Command attribute), 13306, 13362, 13398
reactionner_tag (shinken.objects.host.Host attribute), 13314, 13370, 13406
reactionner_tag (shinken.objects.service.Service attribute), 13330, 13387, 13423
read() (shinken.clients.livestatus.LSSyncConnection method), 13302, 13357, 13393
readable() (shinken.clients.livestatus.LSAsynConnection method), 13302, 13356, 13393
Realm (class in shinken.objects.realm), 13323, 13379, 13415
realm (shinken.objects.host.Host attribute), 13314, 13370, 13406
Realms (class in shinken.objects.realm), 13323, 13380, 13416
reason_type (shinken.notification.Notification attribute), 13296, 13350
rebuild_ref() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13381, 13418
ref (shinken.check.Check attribute), 13284, 13339
register_a_problem() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
register_child() (shinken.objects.host.Host method), 13314, 13370, 13406
register_local_log() (shinken.log.Log method), 13293, 13348
register_son_in_parent_child_dependencies() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418

remove_in_progress_check() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
remove_in_progress_notification() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
remove_in_progress_notifications() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
remove_instance() (shinken.modulesmanager.ModulesManager method), 13294, 13349
remove_templates() (shinken.objects.item.Items method), 13320, 13376, 13412
remove_twins() (shinken.objects.item.Items method), 13320, 13376, 13412
replace_members() (shinken.objects.itemgroup.Itemgroup method), 13320, 13377, 13413
reset_idle() (shinken.worker.Worker method), 13300, 13355
resolve_command() (shinken.macroresolver.MacroResolver method), 13293, 13348
resolve_daterange() (shinken.objects.timeperiod.Timeperiod method), 13335, 13391, 13427
resolve_elements() (shinken.complexexpression.ComplexExpressionNode method), 13285, 13340
resolve_simple_macros_in_string() (shinken.macroresolver.MacroResolver method), 13293, 13348
Resultmodulation (class in shinken.objects.resultmodulation), 13324, 13380, 13416
Resultmodulations (class in shinken.objects.resultmodulation), 13324, 13380, 13417
resultmodulations (shinken.objects.host.Host attribute), 13314, 13370, 13406
resultmodulations (shinken.objects.service.Service attribute), 13330, 13387, 13423
retain_nonstatus_information (shinken.objects.host.Host attribute), 13314, 13370, 13406
retain_nonstatus_information (shinken.objects.service.Service attribute), 13330, 13387, 13423
retain_status_information (shinken.objects.host.Host attribute), 13314, 13370, 13406
retain_status_information (shinken.objects.service.Service attribute), 13330, 13387, 13423
RETENTION (shinken.basemodule.ModulePhases attribute), 13283, 13338
retry_interval (shinken.objects.host.Host attribute), 13314, 13370, 13406
retry_interval (shinken.objects.service.Service attribute), 13330, 13387, 13423
return_code (shinken.objects.host.Host attribute), 13314, 13370, 13406
return_code (shinken.objects.service.Service attribute), 13330, 13387, 13423
run() (in module shinken.clients.LSB), 13301, 13356, 13393
RUNNING (shinken.basemodule.ModulePhases attribute), 13283, 13338
running_properties (shinken.objects.checkmodulation.CheckModulation attribute), 13306, 13361, 13397
running_properties (shinken.objects.contact.Contact attribute), 13307, 13363, 13399
running_properties (shinken.objects.discoveryrule.Discoveryrule attribute), 13308, 13364, 13400
running_properties (shinken.objects.discoveryrun.Discoveryrun attribute), 13309, 13364, 13401
running_properties (shinken.objects.escalation.Escalation attribute), 13309, 13365, 13401
running_properties (shinken.objects.host.Host attribute), 13314, 13370, 13407
running_properties (shinken.objects.item.Item attribute), 13319, 13375, 13411
running_properties (shinken.objects.macromodulation.MacroModulation attribute), 13321, 13377, 13413
running_properties (shinken.objects.notificationway.NotificationWay attribute), 13322, 13378, 13414
running_properties (shinken.objects.pack.Pack attribute), 13322, 13379, 13415
running_properties (shinken.objects.realm.Realm attribute), 13323, 13380, 13416
running_properties (shinken.objects.service.Service attribute), 13330, 13387, 13423
running_properties (shinken.objects.timeperiod.Timeperiod attribute), 13335, 13391, 13428
running_properties (shinken.objects.trigger.Trigger attribute), 13335, 13392, 13428

S

s_time (shinken.check.Check attribute), 13284, 13339
s_time (shinken.eventhandler.EventHandler attribute), 13291, 13346
s_time (shinken.notification.Notification attribute), 13296, 13350
safe_print() (in module shinken.util), 13300, 13354
scatter_notification() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
sched_id (shinken.eventhandler.EventHandler attribute), 13291, 13346
sched_id (shinken.notification.Notification attribute), 13296, 13350
schedule() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
scheduled_downtime_depth (shinken.objects.host.Host attribute), 13314, 13370, 13407

scheduled_downtime_depth (shinken.objects.service.Service attribute), 13330, 13387, 13423
scheduler_no_spare_first() (in module shinken.util), 13300, 13354
SchedulingItem (class in shinken.objects.schedulingitem), 13324, 13381, 13417
send_message() (shinken.worker.Worker method), 13301, 13355
Service (class in shinken.objects.service), 13326, 13382, 13418
service_dependencies (shinken.objects.service.Service attribute), 13330, 13387, 13423
service_description (shinken.notification.Notification attribute), 13296, 13350
service_description (shinken.objects.service.Service attribute), 13330, 13387, 13423
service_description (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13390, 13426
Servicedependencies (class in shinken.objects.servicedependency), 13332, 13388, 13425
Servicedependency (class in shinken.objects.servicedependency), 13332, 13389, 13425
Serviceescalation (class in shinken.objects.serviceescalation), 13332, 13389, 13425
Serviceescalations (class in shinken.objects.serviceescalation), 13333, 13389, 13425
ServiceExtInfo (class in shinken.objects.serviceextinfo), 13333, 13389, 13425
Servicegroup (class in shinken.objects.servicegroup), 13334, 13390, 13426
Servicegroups (class in shinken.objects.servicegroup), 13334, 13390, 13427
servicegroups (shinken.objects.service.Service attribute), 13330, 13387, 13423
Services (class in shinken.objects.service), 13331, 13388, 13424
services (shinken.objects.host.Host attribute), 13314, 13370, 13407
ServicesExtInfo (class in shinken.objects.serviceextinfo), 13333, 13390, 13426
set_exit_handler() (shinken.basemodule.BaseModule method), 13282, 13337
set_human_format() (shinken.log.Log method), 13293, 13348
set_impact_state() (shinken.objects.host.Host method), 13314, 13370, 13407
set_impact_state() (shinken.objects.service.Service method), 13330, 13387, 13423
set_level() (shinken.log.Log method), 13293, 13348
set_loaded_into() (shinken.basemodule.BaseModule method), 13282, 13337
set_max_queue_size() (shinken.modulesmanager.ModulesManager method), 13294, 13349
set_modules() (shinken.modulesmanager.ModulesManager method), 13294, 13349
set_myself_as_problem() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
set_proctitle() (shinken.basemodule.BaseModule method), 13282, 13337
set_proctitle() (shinken.worker.Worker method), 13301, 13355
set_signal_handler() (shinken.basemodule.BaseModule method), 13282, 13337
set_state_from_exit_status() (shinken.objects.host.Host method), 13314, 13370, 13407
set_state_from_exit_status() (shinken.objects.service.Service method), 13331, 13387, 13423
set_to_restart() (shinken.modulesmanager.ModulesManager method), 13294, 13349
set_type_active() (shinken.check.Check method), 13284, 13339
set_type_passive() (shinken.check.Check method), 13284, 13339
set_unreachable() (shinken.objects.host.Host method), 13315, 13370, 13407
set_unreachable() (shinken.objects.service.Service method), 13331, 13387, 13423
set_zombie() (shinken.worker.Worker method), 13301, 13355
setdefault() (shinken.sorteddict.SortedDict method), 13299, 13353
shinken.__init__ (module), 13281, 13336
shinken.acknowledge (module), 13281, 13336
shinken.action (module), 13281, 13336
shinken.autoslots (module), 13281, 13336
shinken.basemodule (module), 13282, 13337
shinken.borg (module), 13283, 13338
shinken.brok (module), 13283, 13338
shinken.check (module), 13283, 13338
shinken.clients (module), 13301, 13356, 13392
shinken.clients.livestatus (module), 13301, 13356, 13393
shinken.clients.LSB (module), 13301, 13356, 13392
shinken.commandcall (module), 13284, 13339
shinken.comment (module), 13285, 13340
shinken.complexexpression (module), 13285, 13340
shinken.contactdowntime (module), 13285, 13340
shinken.daemons (module), 13302, 13358, 13394
shinken.daterange (module), 13286, 13341
shinken.db (module), 13287, 13342
shinken.db_mysql (module), 13288, 13343

shinken.db_oracle (module), 13288, 13343
shinken.db_sqlite (module), 13288, 13343
shinken.dependencynode (module), 13288, 13343
shinken.discovery (module), 13303, 13358, 13394
shinken.dispatcher (module), 13289, 13344
shinken.downtime (module), 13289, 13344
shinken.easter (module), 13289, 13344
shinken.eventhandler (module), 13291, 13346
shinken.graph (module), 13292, 13346
shinken.load (module), 13292, 13347
shinken.log (module), 13292, 13347
shinken.macroresolver (module), 13293, 13348
shinken.memoized (module), 13293, 13348
shinken.message (module), 13294, 13348
shinken.misc (module), 13303, 13358, 13394
shinken.misc.datamanager (module), 13303, 13358, 13394
shinken.misc.filter (module), 13304, 13359, 13395
shinken.misc.md5crypt (module), 13304, 13359, 13396
shinken.misc.perfdata (module), 13304, 13360, 13396
shinken.misc.sorter (module), 13305, 13360, 13396
shinken.misc.termcolor (module), 13305, 13360, 13396
shinken.modulesmanager (module), 13294, 13349
shinken.notification (module), 13295, 13349
shinken.objects (module), 13305, 13360, 13397
shinken.objects.businessimpactmodulation (module), 13305, 13361, 13397
shinken.objects.checkmodulation (module), 13305, 13361, 13397
shinken.objects.command (module), 13306, 13362, 13398
shinken.objects.contact (module), 13307, 13362, 13399
shinken.objects.contactgroup (module), 13307, 13363, 13399
shinken.objects.discoveryrule (module), 13308, 13364, 13400
shinken.objects.discoveryrun (module), 13308, 13364, 13400
shinken.objects.escalation (module), 13309, 13365, 13401
shinken.objects.host (module), 13309, 13365, 13401
shinken.objects.hostdependency (module), 13316, 13372, 13408
shinken.objects.hostescalation (module), 13316, 13372, 13408
shinken.objects.hostextinfo (module), 13317, 13373, 13409
shinken.objects.hostgroup (module), 13317, 13373, 13410
shinken.objects.item (module), 13318, 13374, 13410
shinken.objects.itemgroup (module), 13320, 13376, 13413
shinken.objects.macromodulation (module), 13321, 13377, 13413
shinken.objects.matchingitem (module), 13321, 13377, 13414
shinken.objects.module (module), 13321, 13377, 13414
shinken.objects.notificationway (module), 13322, 13378, 13414
shinken.objects.pack (module), 13322, 13379, 13415
shinken.objects.realm (module), 13323, 13379, 13415
shinken.objects.resultmodulation (module), 13324, 13380, 13416
shinken.objects.schedulingitem (module), 13324, 13381, 13417
shinken.objects.service (module), 13326, 13382, 13418
shinken.objects.servicedependency (module), 13332, 13388, 13425
shinken.objects.serviceescalation (module), 13332, 13389, 13425
shinken.objects.serviceextinfo (module), 13333, 13389, 13425
shinken.objects.servicegroup (module), 13334, 13390, 13426
shinken.objects.timeperiod (module), 13334, 13391, 13427
shinken.objects.trigger (module), 13335, 13392, 13428
shinken.property (module), 13296, 13351
shinken.singleton (module), 13298, 13352
shinken.sorteddict (module), 13298, 13352
shinken.util (module), 13299, 13354
shinken.webui (module), 13336, 13392, 13429
shinken.worker (module), 13300, 13355
should_be_scheduled (shinken.objects.host.Host attribute), 13315, 13370, 13407
should_be_scheduled (shinken.objects.service.Service attribute), 13331, 13387, 13423
SimpleQueryGenerator (class in shinken.clients.LSB), 13301, 13356, 13392
Singleton (class in shinken.singleton), 13298, 13352
smallest_item() (shinken.sorteddict.SortedDict method), 13299, 13353
smallest_key() (shinken.sorteddict.SortedDict method), 13299, 13353
sort_by_ids() (in module shinken.util), 13300, 13354
SortedDict (class in shinken.sorteddict), 13298, 13352
source_problems (shinken.objects.host.Host attribute), 13315, 13370, 13407
source_problems (shinken.objects.service.Service attribute), 13331, 13387, 13423
stack_query() (shinken.clients.livestatus.LSAsynConnection method), 13302, 13356, 13393
stalking_options (shinken.objects.host.Host attribute), 13315, 13370, 13407
stalking_options (shinken.objects.service.Service attribute), 13331, 13387, 13423
StandardDaterange (class in shinken.daterange), 13287, 13342

start() (shinken.basemodule.BaseModule method), 13282, 13337
start() (shinken.worker.Worker method), 13301, 13355
start_external_instances() (shinken.modulesmanager.ModulesManager method), 13294, 13349
start_time (shinken.notification.Notification attribute), 13296, 13350
start_time (shinken.objects.host.Host attribute), 13315, 13371, 13407
start_time (shinken.objects.service.Service attribute), 13331, 13387, 13423
state (shinken.check.Check attribute), 13284, 13339
state (shinken.notification.Notification attribute), 13296, 13351
state (shinken.objects.host.Host attribute), 13315, 13371, 13407
state (shinken.objects.service.Service attribute), 13331, 13387, 13423
state_before_hard_unknown_reach_phase (shinken.objects.host.Host attribute), 13315, 13371, 13407
state_before_hard_unknown_reach_phase (shinken.objects.service.Service attribute), 13331, 13387, 13423
state_before_impact (shinken.objects.host.Host attribute), 13315, 13371, 13407
state_before_impact (shinken.objects.service.Service attribute), 13331, 13387, 13423
state_changed_since_impact (shinken.objects.host.Host attribute), 13315, 13371, 13407
state_changed_since_impact (shinken.objects.service.Service attribute), 13331, 13387, 13423
state_id (shinken.objects.host.Host attribute), 13315, 13371, 13407
state_id (shinken.objects.service.Service attribute), 13331, 13387, 13423
state_id_before_impact (shinken.objects.host.Host attribute), 13315, 13371, 13407
state_id_before_impact (shinken.objects.service.Service attribute), 13331, 13387, 13423
state_type (shinken.objects.host.Host attribute), 13315, 13371, 13407
state_type (shinken.objects.service.Service attribute), 13331, 13387, 13424
state_type_id (shinken.objects.host.Host attribute), 13315, 13371, 13407
state_type_id (shinken.objects.service.Service attribute), 13331, 13387, 13424
status (shinken.check.Check attribute), 13284, 13339
status (shinken.eventhandler.EventHandler attribute), 13291, 13346
status (shinken.notification.Notification attribute), 13296, 13351
statusmap_image (shinken.objects.host.Host attribute), 13315, 13371, 13407
statusmap_image (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
stop_all() (shinken.modulesmanager.ModulesManager method), 13294, 13349
stop_process() (shinken.basemodule.BaseModule method), 13282, 13337
str() (shinken.message.Message method), 13294, 13349
stringify() (shinken.db.DB method), 13287, 13342
StringProp (class in shinken.property), 13297, 13352
strip_and_uniq() (in module shinken.util), 13300, 13354
switch_zeros_of_values() (shinken.dependencynode.DependencyNode method), 13288, 13343

T

t_to_go (shinken.check.Check attribute), 13284, 13339
t_to_go (shinken.eventhandler.EventHandler attribute), 13291, 13346
t_to_go (shinken.notification.Notification attribute), 13296, 13351
tags (shinken.objects.host.Host attribute), 13315, 13371, 13407
tags (shinken.objects.service.Service attribute), 13331, 13387, 13424
terminate() (shinken.worker.Worker method), 13301, 13355
time_to_orphanage (shinken.objects.host.Host attribute), 13315, 13371, 13407
time_to_orphanage (shinken.objects.service.Service attribute), 13331, 13387, 13424
timeout (shinken.commandcall.CommandCall attribute), 13285, 13340
timeout (shinken.eventhandler.EventHandler attribute), 13291, 13346
timeout (shinken.notification.Notification attribute), 13296, 13351
timeout (shinken.objects.command.Command attribute), 13306, 13362, 13398
timeout (shinken.objects.host.Host attribute), 13315, 13371, 13407
timeout (shinken.objects.service.Service attribute), 13331, 13387, 13424
Timeperiod (class in shinken.objects.timeperiod), 13334, 13391, 13427
Timeperiods (class in shinken.objects.timeperiod), 13335, 13391, 13428
Timerange (class in shinken.daterange), 13287, 13342
to64() (in module shinken.misc.md5crypt), 13304, 13360, 13396

to_best_int_float() (in module shinken.util), 13300, 13354
to_bool() (in module shinken.util), 13300, 13354
to_char() (in module shinken.util), 13300, 13354
to_float() (in module shinken.util), 13300, 13354
to_hostnames_list() (in module shinken.util), 13300, 13354
to_int() (in module shinken.util), 13300, 13354
to_list_of_names() (in module shinken.util), 13300, 13355
to_list_string_of_names() (in module shinken.util), 13300, 13355
to_name_if_possible() (in module shinken.util), 13300, 13355
to_split() (in module shinken.util), 13300, 13355
to_svc_hst_distinct_lists() (in module shinken.util), 13300, 13355
topology_change (shinken.objects.host.Host attribute), 13315, 13371, 13407
topology_change (shinken.objects.service.Service attribute), 13331, 13387, 13424
trending_policies (shinken.objects.host.Host attribute), 13315, 13371, 13407
trending_policies (shinken.objects.service.Service attribute), 13331, 13387, 13424
Trigger (class in shinken.objects.trigger), 13335, 13392, 13428
trigger (shinken.objects.host.Host attribute), 13315, 13371, 13407
trigger (shinken.objects.service.Service attribute), 13331, 13387, 13424
trigger_me() (shinken.downtime.Downtime method), 13289, 13344
trigger_name (shinken.objects.host.Host attribute), 13315, 13371, 13407
trigger_name (shinken.objects.service.Service attribute), 13331, 13387, 13424
Triggers (class in shinken.objects.trigger), 13335, 13392, 13428
triggers (shinken.objects.host.Host attribute), 13315, 13371, 13407
triggers (shinken.objects.service.Service attribute), 13331, 13388, 13424
try_instance_init() (shinken.modulesmanager.ModulesManager method), 13294, 13349
try_to_restart_deads() (shinken.modulesmanager.ModulesManager method), 13294, 13349
type (shinken.check.Check attribute), 13284, 13339
type (shinken.eventhandler.EventHandler attribute), 13291, 13346
type (shinken.notification.Notification attribute), 13296, 13351

U

u_time (shinken.check.Check attribute), 13284, 13339
u_time (shinken.eventhandler.EventHandler attribute), 13291, 13346
u_time (shinken.notification.Notification attribute), 13296, 13351
unacknowledge_problem() (shinken.objects.item.Item method), 13319, 13375, 13411
unacknowledge_problem_if_not_sticky() (shinken.objects.item.Item method), 13319, 13375, 13411
unix_md5_crypt() (in module shinken.misc.md5crypt), 13304, 13360, 13396
unset_impact_state() (shinken.objects.host.Host method), 13315, 13371, 13407
unset_impact_state() (shinken.objects.service.Service method), 13331, 13388, 13424
UnusedProp (class in shinken.property), 13296, 13351
update() (shinken.sorteddict.SortedDict method), 13299, 13353
update_business_impact_value() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
update_event_and_problem_id() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
update_flapping() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
update_hard_unknown_phase_state() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
update_in_checking() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
update_load() (shinken.load.Load method), 13292, 13347
update_notification_command() (shinken.objects.schedulingitem.SchedulingItem method), 13325, 13382, 13418
usage() (in module shinken.clients.LSB), 13301, 13356, 13393
use (shinken.objects.command.Command attribute), 13306, 13362, 13398
use (shinken.objects.host.Host attribute), 13315, 13371, 13407
use (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409
use (shinken.objects.service.Service attribute), 13331, 13388, 13424
use (shinken.objects.serviceextinfo.ServiceExtInfo attribute), 13333, 13390, 13426

V

valid (shinken.commandcall.CommandCall attribute), 13285, 13340

values() (shinken.sorteddict.SortedDict method), 13299, 13353
vrml_image (shinken.objects.host.Host attribute), 13315, 13371, 13407
vrml_image (shinken.objects.hostextinfo.HostExtInfo attribute), 13317, 13373, 13409

W

wait_returns() (shinken.clients.livestatus.LSAsynConnection method), 13302, 13357, 13393
want_brok() (shinken.basemodule.BaseModule method), 13282, 13337
want_host_notification() (shinken.objects.contact.Contact method), 13307, 13363, 13399
want_host_notification() (shinken.objects.notificationway.NotificationWay method), 13322, 13378, 13414
want_service_notification() (shinken.objects.contact.Contact method), 13307, 13363, 13399
want_service_notification() (shinken.objects.notificationway.NotificationWay method), 13322, 13378, 13415
WARNING (shinken.log.Log attribute), 13292, 13347
warning() (shinken.log.Log method), 13293, 13348
was_in_hard_unknown_reach_phase (shinken.objects.host.Host attribute), 13315, 13371, 13407
was_in_hard_unknown_reach_phase (shinken.objects.service.Service attribute), 13331, 13388, 13424
WeekDayDaterange (class in shinken.daterange), 13287, 13342
weekdays (shinken.daterange.Daterange attribute), 13286, 13341
work() (shinken.basemodule.BaseModule method), 13282, 13337
work() (shinken.worker.Worker method), 13301, 13355
Worker (class in shinken.worker), 13300, 13355
worker (shinken.check.Check attribute), 13284, 13339
worker (shinken.eventhandler.EventHandler attribute), 13291, 13346
worker (shinken.notification.Notification attribute), 13296, 13351
worse_first() (in module shinken.misc.sorter), 13305, 13360, 13396
writable() (shinken.clients.livestatus.LSAsynConnection method), 13302, 13357, 13393